Intergiciel agent pour le déploiement et la configuration d'applications distribuées dans des environnements ambiants

Ferdinand Piette

THÈSE DE DOCTORAT DE
l'UNIVERSITÉ PIERRE ET MARIE CURIE

Spécialité

Informatique

École doctorale Informatique, Télécommunications et Électronique (Paris)

Présentée par

Ferdinand PIETTE

Pour obtenir le grade de

DOCTEUR de l'UNIVERSITÉ PIERRE ET MARIE CURIE
Sorbonne université, UPMC Univ Paris 06

Sujet de la thèse :

« Intergiciel Agent pour le Déploiement et la Configuration d'Applications Distribuées dans des Environnements Ambiants »

Soutenue le 17 janvier 2017

Devant le jury composé de :

Mme Amal El Fallah Seghrouchni	Directrice de thèse
M. Cédric Dinont	Encadrant
M. Stéphane Galland	Rapporteur
Mme Salima Hassas	Rapporteur
M. Rémy Courdier	Examinateur
M. Jean-Michel Ilié	Examinateur
M. René Mandiau	Examinateur
M. Patrick Taillibert	Invité

Table des matières

Résumé

L'évolution des technologies de l'information ainsi que la miniaturisation constante des composants électroniques de ces dernières décennies ont permis de doter les objets de la vie de tous les jours de capacités de calcul et de communication. Ces « objets connectés » sont disséminés dans l'environnement de l'utilisateur et offrent à celui-ci un ensemble de services visant à l'assister dès que celui-ci en a besoin. L'Intelligence Ambiante (IAm) est une vision dans laquelle ces objets connectés coopèrent les uns avec les autres afin de fournir à l'utilisateur des services encore plus intelligents de manière totalement transparente et non intrusive.

Pour que cette vision devienne réalité, il est indispensable de considérer plusieurs aspects des environnements ambiants. Tout d'abord, ceux-ci sont composés d'entités hétérogènes, aussi bien en puissance de calcul, en fonctionnalités, mais aussi au niveau de leurs contraintes (énergétiques, de mobilité, de puissance de calcul, etc.) et de leurs protocoles. Ensuite, les intégrations dites « verticales » (les données des capteurs sont externalisées sur les serveurs d'une entreprise) permettent certes une interopérabilité plus importante, mais engendrent des problèmes de saturation des canaux de communication dus à la quantité d'informations toujours plus importante qui est générée, ainsi que des questionnements sur la sécurité et la confidentialité de ces informations. Pour pallier ces problèmes, les intégrations dites « horizontales », dans lesquelles les entités matérielles sont mises en relation directement au sein de l'infrastructure, sont encouragées. Enfin, le nombre grandissant d'objets connectés pose le problème du passage à l'échelle ainsi que de la dynamicité de ces environnements.

Dans cette thèse, nous adressons le problème du déploiement et de la configuration automatique d'applications au sein de tels environnements ambiants. Nous proposons des mécanismes permettant la sélection et la configuration d'entités matérielles de l'environnement qui supporteront l'exécution des applications.

Nous proposons un intergiciel basé sur le paradigme Multi-Agents. À partir d'une description de l'environnement ambiant – c'est-à-dire des entités matérielles disponibles dans l'infrastructure, de leurs propriétés et de leurs relations, mais aussi des applications à déployer ainsi que de leurs besoins et de leurs contraintes – les différents agents logiciels du système collaborent afin de sélectionner les entités de l'infrastructure qui serviront de support à l'exécution des applications, respectant leurs besoins et leurs contraintes.

Nous proposons dans ce travail (1) une méta-modélisation permettant de décrire l'environnement, (2) une formalisation mathématique ainsi qu'un algorithme distribué permettant de sélectionner les entités matérielles à utiliser pour déployer une application tout en conservant la confidentialité des informations relatives à la structure et aux propriétés de l'infrastructure matérielle, ainsi qu' (3) un système multi-agent encapsulant ces mécanismes, proposant une solution distribuée au déploiement d'applications et améliorant la confidentialité des ressources. Une implémentation de la solution est aussi proposée et des résultats d'expérimentations permettent de valider le fonctionnement et les performances de l'algorithme dans le cadre de la gestion d'habitations.

Remerciements

Le résultat de cette thèse est le fruit d'une longue réflexion de plus de cinquante mois consacrés presque exclusivement à ce travail. Durant ce périple, tout n'a pas été linéaire, mais semé d'embûches : entre obsessions qui vous réveillent la nuit, coups de blues lorsque l'on doit jeter des semaines de travaux qui n'ont pas aboutis, stress lorsque l'on voit se rapprocher les échéances ou lorsque l'on navigue dans le brouillard, mais aussi joie quand un ensemble flou d'idées cristallise soudain, donnant naissance à un embryon de solution, ou encore accomplissement lorsque l'on présente les résultats de ses travaux en conférence.

Bref, cette réflexion mouvementée et passionnante n'aurait pas été possible sans l'aide d'un grand nombre de personnes qui m'ont, à un moment ou à un autre, et parfois sans le savoir, apporté leur soutien par leurs échanges, discussions, confrontations, écoute, ou tout simplement par leur présence. C'est pourquoi je tiens à remercier chaleureusement ces autres acteurs de cette thèse.

TODO...

Avant-propos

> Dans les conditions normales, le chercheur n'est pas un novateur, mais quelqu'un qui résout des problèmes, et les problèmes sur lesquels il se concentre sont justement ceux dont il pense qu'ils peuvent être énoncés et résolus à l'intérieur de la tradition scientifique existante.
>
> *La structure des révolutions scientifiques* - Thomas S. Kuhn

Lorsque j'ai débuté ce travail de thèse, je m'imaginais, naïvement, je l'admets volontiers, révolutionner le monde en proposant des concepts novateurs qui allaient changer la vision que l'on porte sur les sciences. Ce n'est qu'au fur et à mesure de ces trois années que j'ai compris que le travail d'un chercheur n'était pas de construire un bâtiment entier à lui tout seul, mais de s'approprier un domaine de recherche : ses travaux, son vocabulaire. Une fois tout cela maitrisé, celui-ci peut alors apporter sa pierre à l'édifice, qu'elle soit simplement d'ordre cosmétique ou, pour les plus chanceux/entreprenant, qu'elle consolide légèrement le bâtiment.

Néanmoins, s'approprier un domaine de recherche, c'est s'imposer un vocabulaire, une façon de penser, en somme toute une série de règles qui bride notre créativité et réduit considérablement notre liberté d'innovation. Si en plus de cela, le jeune chercheur se cloisonne à son domaine de recherche sans se préoccuper des autres domaines qui peuvent pourtant abriter des problèmes similaires, mais énoncés et traités sous un autre angle, cela engendre une stérilité au niveau des idées scientifiques.

Un juste milieu entre une étude approfondie d'un domaine de recherche et une étude transversale de plusieurs autres domaines permet de stimuler sa créativité et de se positionner sur des problèmes qui n'ont pas encore été énoncés par les domaines concernés. La recherche est pour moi un subtil mélange entre rigueur, rationalité, mais aussi créativité et intuition.

Le travail de thèse est pour la plupart le premier vrai pas dans le monde de la recherche. Le manuscrit rédigé à l'issue de ce travail est donc là pour prouver que l'on a su s'approprier un domaine de recherche mais aussi que l'on a déjà réussi à acquérir suffisamment de recul pour isoler certains problèmes et proposer une ou des solutions.

C'est donc dans cette optique que je vous présente ce manuscrit de thèse de doctorat intitulé : « Intergiciel Agent pour le Déploiement et la Configuration d'Applications Distribuées dans des Environnements Ambiants » . Ce travail se situe à l'intersection des domaines de recherche de l'intelligence ambiante, de l'internet des objets et des systèmes distribués, notamment des systèmes multi-agents. D'autres domaines annexes ont aussi été une source d'inspiration. C'est le cas du Cloud Computing, de l'utilisation et de l'évolution des techniques de Big Data, du développement commercial des objets connectés ainsi que de la tendance actuelle des technologies d'ingénierie informatique.

Introduction

Motivations

Des domaines de recherche comme l'Intelligence Ambiante ou l'Internet des Objets sont apparus au début des années 2000 grâce aux innovations technologiques et à la miniaturisation continue des composants électroniques. Ces périphériques électroniques et informatiques peuvent échanger des données de plus en plus facilement, ce qui conduit à de nouveaux domaines applicatifs comme les Villes Intelligentes, dont une des caractéristiques principales est sa forte hétérogénéité, aussi bien matérielle que logicielle. En effet, le matériel est sans cesse plus puissant : il faut maintenant prendre en compte non seulement des ordinateurs, mais aussi des smartphones, des micro-ordinateurs intégrés dans différents types de périphériques comme des box télévisions, ainsi que d'autres capteurs et objets connectés. Les moyens et les protocoles de communication sont aussi très spécifiques à un usage : certaines applications utilisent du Wifi afin de bénéficier d'une bande passante importante sur une courte distance, alors que d'autres applications utilisent des protocoles comme SigFox ou LoRa pour envoyer très peu de données, mais sur de très longues distances, et avec une très faible consommation énergétique. La vitesse d'acquisition, la quantité ainsi que la sémantique d'un flux de données peuvent aussi être très variables et tendre à augmenter de manière importante afin de fournir des services et des applications toujours plus réactives et avec une forte plus-value : les données de consommation électrique peuvent être acquises une fois par mois pour un service de facturation, dix fois par heure pour une application de chauffage, et toutes les secondes pour une application de monitoring d'activité. D'après une étude menée par l'IDATE, le marché de ces périphériques électroniques, ou « objets connectés », atteindra d'ici 2020 plus de 50 milliards d'objets actifs [Evans 11].

Les travaux de thèse que nous allons présenter dans ce mémoire ont été effectués, au départ, dans la perspective de simplifier le déploiement d'applications distribuées au sein d'environnements dynamiques. Le déploiement d'applications sur une plateforme ambiante doit être aussi facile que le téléchargement d'une application sur un smartphone depuis un store d'applications. Pour atteindre ce niveau de « facilité », il nous fallait, dans un premier temps, décorréler la partie matérielle de la partie logicielle. Il nous semblait dès lors intéressant de décorréler ces parties en décrivant, à différents niveaux, les nombreuses entités entrant en jeu. Une fois les spécificités et les propriétés de ces entités modélisées, il a été possible de proposer des mécanismes permettant le

déploiement d'applications sur une infrastructure matérielle représentant l'environnement dynamique. Dans un second temps, les systèmes multi-agents nous sont apparus comme étant une approche intéressante pour concevoir un système applicable dans des environnements réels. L'approche multi-agents vise à distribuer la résolution d'un problème à différentes entités logicielles autonomes, appelées agents, qui vont interagir, coopérer et/ou négocier entre elles. D'une part, la modularité qu'apporte ce paradigme nous apparaît comme intéressante pour privilégier des raisonnements locaux, isoler plus facilement les différentes couches du système et faciliter le passage à l'échelle, et d'autre part, l'autonomie que l'on accorde aux agents laissait sous-entendre à la fois une aisance de conception, mais aussi une certaine robustesse. Cette manière d'appréhender les choses nous a permis de nous focaliser notamment sur un point clef des systèmes d'information de nos jours : la confidentialité des informations.

Contexte

Cette thèse, intitulée « Intergiciel Agent pour le Déploiement et la Configuration d'Applications Distribuées dans des Environnements Ambiants » est dirigée par Mme Amal El Fallah Seghrouchni et co-dirigé par M. Patrick Taillibert du LIP6 (Laboratoire d'Informatique de Paris VI) et par M. Cédric Dinont de l'ISEN Lille (Institut Supérieur de l'Électronique et du Numérique). Elle est co-financée par Lille Métropole Communauté Urbaine (LMCU), au travers de l'Université Catholique de Lille (UCL), ainsi que par le projet européen INTERREG i-stay@home [1]. Ce projet est porté par des bailleurs sociaux et inclut divers partenaires techniques, dont l'ISEN. Il a pour but de créer une plateforme transnationale d'intégration et de déploiement de services à destination des populations fragiles. Ce travail de thèse, vis-à-vis du projet i-stay@home, est de fournir à cette plate-forme les spécifications pour l'intégration de mécanismes de haut niveau permettant le déploiement automatique et la configuration d'applications dans des environnements intelligents de type logements sociaux.

La partie théorique se focalise sur des problématiques de déploiement et de configuration automatique d'entités logicielles, composantes d'applications, sur du matériel très hétérogène disponible au sein d'environnements dits ambiants. Ansi, l'infrastructure de tels environnements est composée d'un ensemble d'applications distribuées utilisant les capteurs et les entités matérielles disséminées dans les logements. Ces entités matérielles sont ajoutées ou retirées au sein d'un logement selon l'envie de son occupant. Un ensemble de caractéristiques a pu être isolé grâce au travail avec les bailleurs du projet i-stay@home. La première caractéristique des environnements ambiants est leur **hétérogénéité**. En effet, les entités matérielles, achetées dans le commerce par le locataire du logement, utilisent des technologies très hétérogènes, aussi bien en terme de logiciels que de protocoles de communication. La seconde caractéristique est la **dynamicité** de ces environnements. Un locataire doit non seulement pouvoir rajouter ou supprimer des entités matérielles au sein de son environnement, mais plus encore, le déploiement des applications doit pouvoir s'adapter à ces changements. La troisième caractéristique est son extensibilité : il est possible de rajouter des parties entières d'infrastructure, parfois à plusieurs ordres de grandeur, ce qui implique que le système

1. http://www.i-stay-home.eu/

qui le prend en charge doit pouvoir faire le **passage à l'échelle**. Le projet étant pris en charge par des bailleurs sociaux, ceux-ci souhaitent pouvoir ajouter de nouveaux logements ainsi que de nouveaux bâtiments au système de manière intuitive et sans baisse de performance afin de créer un réseau de logements et de bâtiments pouvant communiquer ensemble. Il nous est alors apparu indispensable que notre proposition s'inspire des travaux du domaine des systèmes distribués. Enfin, la dernière caractéristique est de permettre de garantir au locataire d'un logement la **confidentialité** de ses ressources. Le locataire doit avoir un contrôle total des ressources qu'il souhaite partager ou, au contraire, garder privées. Il nous a semblé important de proposer une solution dans laquelle les bailleurs n'auraient pas le moyen de récupérer les informations sur l'infrastructure matérielle disponible dans les différents logements.

Cette thèse possède aussi une composante d'ingénierie : les résultats théoriques de mes recherches sont en cours de mise en œuvre au sein d'une réplique de logement intelligent à Urbawood, sur le site d'Euratechnologies, à Lille. Ce démonstrateur permettra de valider les concepts présentés dans ce travail et pourra servir de tremplin vers une mise en œuvre au sein de logements réels. Les aspects sociétaux en jeu entre dans l'axe prioritaire « Innover sur des axes stratégiques en réponse aux changements du monde » du thème « Handicap, dépendance, citoyenneté » de la politique de financement des thèses UCL par LMCU.

Objectifs

Le but de ce travail de thèse est de faciliter le déploiement d'applications dans des environnements ambiants. Le cas d'usage global se focalise sur un utilisateur qui souhaite avoir une application de portier vidéo pour son logement. Une fois que cet utilisateur a sélectionné, à partir d'un store d'applications, l'application qu'il souhaite déployer, le système se charge alors de sélectionner dans l'environnement les entités matérielles dont l'application en question a besoin. Une fois que ces entités matérielles ont été choisies, le système peut alors configurer l'environnement et déployer l'application.

Afin de proposer une solution prenant en compte toutes les caractéristiques des environnements ambiants (hétérogénéité, dynamicité, passage à l'échelle et confidentialité), nous avons commencé par faire l'étude des systèmes existants permettant de gérer des environnements ambiants. L'approche multi-agents, qui vise à distribuer la résolution d'un problème entre différents agents autonomes interagissant entre eux, est déjà très utilisée en Intelligence Ambiante et l'apport des travaux existants se focalise, entres autre, sur le raisonnement au niveau des données et des événements transitant dans le système. Bien souvent, un agent gère une entité matérielle ou un lieu physique et échange des données et des événements avec les autres agents du système. Cette façon de faire permet de raisonner sur le contexte environnemental et celui de l'utilisateur. Néanmoins les parties logicielle et matérielle s'en trouvent fortement corrélées : les applications ambiantes sont alors basées sur un système multi-agents et doivent être développées en utilisant les technologies mis en avant par le SMA.

Les problématiques de déploiement automatique ont déjà été traitées dans la communauté travaillant sur le calcul distribué. Néanmoins, ces travaux se focalisent sur les

cas où le matériel sur lequel peuvent être déployées les entités logicielles est homogène : par exemple, un cluster de serveurs dans le cas des applications de BigData ou de Cloud Computing. En intelligence ambiante, le matériel disponible dans les logements ou chez le bailleur est très hétérogène : cela peut aller d'un cluster de serveurs hébergé par le bailleur à un simple micro-contrôleur ou encore à une « box Internet » disponible chez le locataire lui-même, en passant par du matériel mobile comme le smartphone du locataire, ainsi que des capteurs et des actionneurs avec peu de ressources. L'on remarque donc que les contraintes de capacité de calcul, de moyen de communication, de mobilité, de robustesse et d'infrastructure logicielle doivent être prises en compte si l'on souhaite tirer parti au mieux de la puissance de calcul omniprésente dans l'environnement de l'utilisateur.

Tous ces travaux, bien qu'incomplets pour remplir nos objectifs, nous ont permis de proposer une solution regroupant les points forts des méthodes étudiées. Notre approche vise à prendre le problème du déploiement d'applications dans un environnement ambiant selon trois aspects. Le premier aspect est celui de la description de l'environnement : l'infrastructure matérielle ainsi que les applications à déployer. Ces descriptions sont établies à l'aide de techniques de représentation des connaissances qui fournissent un cadre formel pour raisonner sur ces connaissances. Nous proposons ainsi un métamodèle permettant de modéliser ces environnements. Le second aspect est celui du raisonnement. À partir de la modélisation de l'environnement, il est possible, grâce à des méthodes de raisonnement, de trouver au sein de l'infrastructure matérielle disponible, les entités qui permettront de déployer une application donnée. Nous proposons ainsi, basé sur une formalisation mathématique, un algorithme permettant de dresser un plan de déploiement, appelé projection, d'une application sur l'infrastructure matérielle. Cet algorithme, dans sa version décentralisée permet de prendre en compte la confidentialité des informations contenues dans le modèle de l'infrastructure matérielle. Enfin, le dernier aspect a été la proposition d'un système distribué, basé sur le paradigme multi-agents, encapsulant ces mécanismes et fournissant un cadre concret pour le développement d'un intergiciel complet (modélisation, raisonnement, déploiement, supervision, confidentialité).

Les objectifs principaux de ce travail de thèse sont donc :

— la prise en compte de l'hétérogénéité des environnements ambiants par la proposition d'un métamodèle permettant de modéliser ces environnements : les entités matérielles disponibles, leurs propriétés, leurs interactions ainsi que les besoins des applications à déployer ;
— la proposition d'un formalisme permettant de raisonner sur les modèles de l'environnement afin de récupérer les entités matérielles à utiliser pour déployer une application donnée ;
— la proposition d'un algorithme distribué basé sur ce formalisme permettant de prendre en compte la confidentialité des informations relatives à l'infrastructure matérielle disponible ;
— la prise en compte du caractère distribué et dynamique des environnements par la proposition d'un système multi-agents permettant une séparation claire des différents partis et autorisant l'introduction de mécanismes suplémentaires pour la confidentialité des ressources de l'utilisateur ;
— enfin, l'implémentation d'un intergiciel mettant en œuvre tous les points ci-

dessus et permettant ainsi d'évaluer le système.

Structure de la thèse

Ce manuscrit de thèse débute par une partie contextuelle dans laquelle nous présentons les problématiques que nous avons choisies de traiter, ainsi que les travaux que nous avons identifiés comme étant les plus pertinents et qui essayent de répondre à ces problématiques. Cette analyse des problématiques et des travaux associés vont nous permettre d'introduire la partie de la thèse concernant mes contributions. Celle-ci a pour but de répondre aux problématiques isolées dans l'état de l'art en proposant des solutions comblant les manques repérés dans les travaux associés. Enfin, la dernière partie se concentre sur l'implémentation et la mise en œuvre d'un intergiciel intégrant les solutions proposées dans la partie contribution, ainsi que sur la présentation de résultats d'expérimentations.

Le Chapitre 1 rend compte du contexte technologique et industriel dans lequel nous vivons. Il explique l'évolution de la manière d'appréhender et de traiter les données dans les systèmes distribués et isole les problématiques d'ingénierie et de recherche que nous souhaitons adresser. Les questions du déploiement d'applications, de la communication horizontale d'objets connectés ainsi que les questions à propos de la sécurité des informations et des ressources y sont évoquées.

Le Chapitre 2 explore les différents travaux de recherche qui visent, de près ou de loin, à répondre aux problématiques évoquées dans le premier chapitre. Nous avons isolé les manques de ces différentes approches, mais aussi les idées que nous jugeons intéressantes qui nous serviront de base pour l'élaboration des solutions proposées dans la partie contribution. Nous commençons par présenter des applications ainsi que des intergiciels pour la gestion d'environnements ambiants. Chacun de ces travaux se focalise souvent sur un point particulier comme la gestion du contexte ou encore la confidentialité des informations. Nous poursuivons avec une étude des travaux sur le déploiement et la configuration automatique d'applications dans les différents domaines des services, du calcul sur grille ainsi et du Cloud Computing. Nous finissons ce chapitre par une présentation de différents formalismes de représentation des connaissances ainsi que des mécanismes de raisonnement associés.

L'analyse de l'état de l'art réalisé dans cette première partie du manuscrit de thèse nous permet de comprendre les problèmes liés au déploiement d'applications ainsi qu'à la gestion des environnements ambiants et de voir comment certains de ces problèmes ont été traités au travers de divers travaux de recherche. Dans la seconde partie, nous proposons une solution propre au déploiement d'applications dans un environnement ambiant.

Dans le Chapitre 3, nous introduisons un métamodèle permettant de modéliser des environnements ambiants (infrastructure et applications) à l'aide de graphes. Cette modélisation permet de décrire les différentes entités du systèmes, ainsi que leurs relations et leurs propriétés. Nous intégrons également cette modélisation dans un formalisme mathématique présenté au Chapitre 4 et permettant d'utiliser l'homomorphisme

de graphe comme outil de projection d'une application sur l'infrastructure matérielle. Enfin, nous proposons au Chapitre 5 un algorithme de matching de graphe distribué permettant de trouver ces homomorphismes représentant les plans de déploiement d'applications. Nous nous focaliserons sur la préservation de la confidentialité des informations relatives à la structure et aux propriétés de l'infrastructure matérielle.

Le Chapitre 6 introduit un système multi-agents prenant en charge le déploiement d'applications. Les mécanismes proposés dans les chapitres précédents sont encapsulés au sein d'agents logiciels coopérant entre-eux afin de déployer les applications sur l'infrastructure. Ce SMA permet de proposer des solutions élégantes pour garantir la sécurité des ressources et le caractère privé des informations relatives à l'utilisateur et à son environnement. Nous présentons dans ce chapitre le fonctionnement d'un agent, la manière dont ils coopèrent entre eux, ainsi que la façon de les grouper et de les organiser.

Cette seconde partie du manuscrit de thèse pose donc les mécanismes abstraits pour le déploiement distribué d'applications sur une infrastructure matérielle existante. Ces mécanismes sont implémentés au sein d'un intergiciel qui est détaillé dans la dernière partie. Cet intergiciel permettra de mettre en œuvre divers scénarios dans le domaine applicatif de la maison connectée au Chapitre 7 afin de valider les concepts théoriques présentés dans la seconde partie. Nous détaillons notamment l'implémentation de l'intergiciel de déploiement d'applications, ainsi que différents résultats d'expérimentations évaluant l'algorithme de projection.

Ce manuscrit de thèse se termine par une conclusion résumant les différentes contributions de ce travail, ainsi que les perspectives à court, moyen et long terme que l'on peut envisager pour les travaux futurs.

Première partie

Contexte

Chapitre 1

Définition des problèmes

> Les humains avaient créé des machines intelligentes afin de disposer de systèmes de réflexes secondaires pour transférer leurs décisions les plus simples à des serviteurs mécaniques. Mais graduellement, les créateurs finirent par ne se laisser aucun rôle. Ils commencèrent à s'aliéner, se déshumaniser, et se laissèrent manipuler. À terme, ils ne furent plus que des robots incapables de décider, de comprendre même le sens de leur existence naturelle.
>
> *Dune, la génèse - La guerre des machines* - Brian Herbert

Sommaire

L'objectif de ce chapitre est de placer le problème de la gestion des environnements intelligents dans le contexte scientifique et industriel actuel. La première Section examine la manière dont les environnements intelligents sont considérés dans le domaine de l'Intelligence Ambiante ainsi que dans celui de l'Internet des Objets (Section 1.1). La seconde section quant à elle isole trois problématiques qui sont un frein au développement industriel des technologies liées à la gestion des environnements intelligents (Section 1.2).

1.1 La gestion des environnements intelligents

Depuis l'invention de l'ordinateur, certains ont espéré pouvoir utiliser la puissance de calcul afin de gérer automatiquement certaines tâches de la vie quotidienne et ainsi de fournir aux Hommes une aide pour faciliter leur prise de décision. On retrouve cette idée fixe dans de nombreux ouvrages de science-fiction dans la litérature ainsi que dans le monde du cinéma qui est personnifiée par le terme d'« Intelligence Artificielle ». Par exemple, le film de Stanley Kubric intitulé « 2001 : l'Odysée de l'espace » nous permet de suivre les péripéties de HAL5000, une intelligence artificielle embarquée à bord d'un vaisseau d'exploration. De même, Brian Herbert, dans sa série de romans intitulée « Dune, La génèse », présente un univers dans lequel les machines ont pris le contrôle de plusieurs milliers de planètes de la galaxie, formant ainsi un réseau gigantesque de machines synchronisées. Ce réseau est personnifié par une super entité du nom d'Omnuim qui a pour unique but d'étendre son empire et d'asservir l'Humanité. Ces œuvres, bien que personnifiant sous la forme d'IA ces systèmes, nous proposent une vision du monde dans laquelle l'informatique gère une grande partie des périphériques environnants. Ils offrent aux humains un ensemble de services visant à leur apporter un niveau de confort supplémentaire (ou visant à les asservir et à les contrôler lorsque la technologie leur échappe).

Le développement de l'informatique et de l'Internet au début des années 1990 a permis de rendre plus concrète cette vision du monde au travers des travaux scientifiques du domaine de l'informatique ubiquitaire (ubiquitous computing [Weiser 93]). Ce domaine de recherche promeut l'utilisation des entités électroniques omniprésents dans l'environnement, comme les téléphones portables ou les ordinateurs, pour faciliter à l'utilisateur l'accès à l'information. Les premiers balbutiements ont permis de poser les bases qui ont servi à l'Intelligence Ambiante, puis à l'Internet des objets de se développer à partir des années 2000.

1.1.1 La vision de l'Intelligence Ambiante

L'Intelligence Ambiante (IAm) est née au début des années 2000 grâce à l'évolution technologique et la miniaturisation continue des périphériques électroniques. Dans la continuité de l'informatique ubiquitaire, le IST Advising Group définit l'Intelligence Ambiante comme une vision de la société de l'information où les personnes sont entourées par des interfaces intuitives et intelligentes qui sont embarquées dans tous types d'objets et d'un environnement qui est capable de reconnaître et de répondre à la

présence de différents individus de manière transparente, non intrusive et souvent invisible [Ducatel 01]. Les applications ambiantes doivent prendre en compte le contexte de l'utilisateur et sont caractérisées par une haute dynamicité. L'Intelligence Ambiante est donc la troisième vague du développement de l'informatique, après les « mainframes » et les ordinateurs personnels. La puissance informatique est maintenant intégrée à l'environnement et embarquée dans des objets de la vie de tous les jours, ce qui la rend invisible vis à vis de l'utilisateur tout en permettant d'assister ce dernier dans sa prise de décision.

Beaucoup de travaux de recherche en Intelligence Ambiante se focalisent sur l'amélioration des interactions humaines avec les applications intelligentes. Ces améliorations sont rendues possibles par la proposition de « frameworks » et de plateformes qui facilitent le développement d'applications context-aware et dynamiques. Ces plateformes offrent des mécanismes pour construire ce type d'applications context-aware en prenant en charge les données et les événements [Hellenschmidt 04] [Johanson 02] ou en encapsulant les entités matérielles et logicielles dans des agents [Chen 04] [El Fallah-Seghrouchni 10]. Les systèmes multi-agents sont souvent utilisés dans ce type de plateformes car ils offrent de bonnes propriétés ainsi que des solutions pour augmenter le niveau d'autonomie, gérer la prise en compte du contexte, améliorer la robustesse, la tolérance aux fautes ou le passage à l'échelle. Néanmoins, tous ces travaux supposent qu'une infrastructure matérielle ubiquitaire sous-jacente existe déjà [O'Hare 12].

D'autres plateformes et intergiciels pour l'Intelligence Ambiante ont été développées pour permettre le développement d'applications sans restriction en terme d'infrastructure matérielle et logicielle sous-jacente, ni en terme de ressources prises en charge [Georgantas 10]. Ces intergiciels utilisent pour la plupart des architectures orientées services et autorisent l'intégration de technologies orientées services [Stavropoulos 13]. Ces architectures utilisent les ressources disponibles présentes dans le système. Les périphériques connectés et les applications hébergées sont abstraits dans des services qui sont ensuite composés et/ou orchestrés par l'intergiciel, en fonction des attentes de l'utilisateur. Tous ces services sont déjà déployés et disponibles. Aucun raisonnement n'est fait sur la localisation de ces ressources : un service de traitement d'images sera hébergé sur une machine spécifique. L'utilisateur pourra utiliser ce service particulier pour traiter ses propres images, mais indépendamment des contraintes de bande passante entre l'entité matérielle hébergeant le service et le client.

1.1.2 La vision de l'Internet des Objets

L'Intelligence Ambiante se focalise sur l'amélioration des interactions humaines avec les applications en faisant l'hypothèse qu'une infrastructure ubiquitaire sous-jacente existe déjà. Dans le même temps, l'Internet des Objets (IoT) vise à fournir cette infrastructure globale pour la société de l'information, autorisant la conception de services avancés en interconnectant des entités (physiques comme virtuelles) grâce à des technologies de télécommunication existantes, évolutives et interopérables [ITU-T 12].

Ces dernières années, l'attention des chercheurs et des industriels s'est portée sur l'IoT qui est considéré comme le futur d'Internet : son extension au monde des objets

physiques. Dans cette vision du monde, les entités matérielles disséminées dans l'environnement seront accessible au travers de l'Internet et constitueront l'ensemble des « objets connectés ». Ces objets connectés, accessibles à l'aide d'un identifiant unique seront alors capables de communiquer et d'échanger des données.

Le développement de l'Internet des Objets implique donc un certain nombre de problèmes à prendre en compte relatifs à l'infrastructure, aux communications, aux interfaces, aux protocoles et aux différents standards. Li et Al. [Li 15] définissent le cadre de la recherche sur les objets connectés en cinq pistes d'exploration :

— la conception d'architectures décentralisées pour la gestion des objets connectés ;
— la définition de mécanismes de confidentialité des informations transitant sur les réseaux ;
— la réalisation de réseaux sécurisés et fiables ;
— le développement de standards ;
— et l'exploration de nouvelles technologies comme les MEMS [1] pour la conception de nouveaux objets connectés

L'émergence de technologies telles que les réseaux de capteurs ou le Cloud Computing rend possible la configuration de réseaux d'entités électroniques qui permettent aux chercheurs de se focaliser sur l'amélioration des interactions entre ces entités matérielles, leur environnement et les utilisateurs. Dans une vision à long terme, tout ce réseau d'objets connectés sera flexible, intelligent, autonome et permettra une fusion entre le monde physique et l'Internet, fournissant à l'utilisateur connectivité et intelligence où qu'il se trouve [Pretz 13].

1.2 Problématiques

1.2.1 Le problème de l'hétérogénéité

Comme nous venons de le voir, l'informatique ubiquitaire et ses extensions (l'Intelligence Ambiante et l'Internet des Objets) permettent d'isoler un certain nombre de caractéristiques des environnements intelligents, et notamment leur dynamicité et leur hétérogénéité. Et même si certains travaux visent à proposer des intergiciels permettant une interopérabilité des technologies, force est de constater que dans le milieu industriel, peu d'acteurs jouent le jeu de l'interopérabilité et chacun redéfinit ses propres technologies et protocoles.

Parmi les succès commerciaux des technologies de l'information de ces vingt cinq dernières années, nous pouvons prendre pour exemple les smartphones. Dans les années 1990, la téléphonie mobile adoptait un modèle très fermé. Chaque constructeur développait ses propres « firmwares » et applications pour les téléphones portables qu'il commercialisait. Après les années 2000 sont apparus les premiers systèmes d'exploitation ouverts (iOS, Android, etc.). Ces systèmes d'exploitation autorisent les développeurs à concevoir des applications utilisant les différentes ressources et périphériques des téléphones (GPS, caméra, accéléromètre, etc.) sans se soucier du modèle du téléphone. Le déploiement de ces applications a aussi été grandement simplifié : un clic

1. Micro Electro-Mechanical Systems

depuis le store d'applications et c'est installé.

Dans la continuité de cette innovation, le « Cloud Computing » permet d'accéder à ses données de n'importe où et sur n'importe quel type de périphérique. Cette technologie a été très vite utilisée par les développeurs d'applications mobiles. Ainsi, le smartphone est donc devenu le premier objet connecté massivement utilisé, capable de synchroniser ses données avec l'Internet.

La multitude des architectures et des protocoles de communication font de la conception d'applications un problème complexe. Mais c'est aussi un atout incontestable pour fournir des systèmes et des applications à hautes performances, pour peu qu'il existe des mécanismes capables de gérer cette hétérogénéité. Bien qu'il existe des mécanismes pour concevoir des applications intelligentes, ainsi que pour gérer des infrastructures matérielles, rien ne permet de sélectionner les entités matérielles de l'infrastructure respectant les contraintes de fonctionnement des applications ; rien ne permet le déploiement automatique de ces dernières.

1.2.2 Le problème du traitement et du transport des données

1.2.2.1 Utilisation actuelle des objets connectés

D'après des études menées par l'IDATE, Gartner ou encore Cisco, il y aura plus de 50 milliards d'objets connectés d'ici 2020 contre seulement 4 milliards en 2011 [Evans 11, Rivera 11]. Ce qui fait des objets connectés la prochaine révolution économique.

La plupart des objets connectés actuels sont capables de produire des données et de les envoyer, via l'Internet, à des serveurs centralisés ou à des « clusters » de serveurs, le plus souvent appartenant à l'entreprise commercialisant le périphérique. Le traitement de ces données est fait par ces serveurs dans le but de fournir des services ou des applications spécifiques aux clients. Les objets connectés et les clusters de serveurs externes communiquent donc de manière verticale. Ces objets ne sont pas encore capables de communiquer directement entre eux et de s'échanger des données de manière horizontale. Nous pouvons noter des similarités avec le monde de la téléphonie mobile des années 90. En effet, il est rare de trouver des objets connectés ouverts qui favorisent la collecte directe des données qu'elle produit, sans passer par un serveur centralisé. Ceci est dû entre autres à la forte hétérogénéité du matériel et des protocoles ainsi qu'au manque de mécanismes capables de la prendre en compte.

Les objets connectés peuvent utiliser des technologies et des protocoles de communication hétérogènes comme le Wifi, le Bluetooth, le ZWave, etc. Tous ces protocoles ont bien entendu des avantages et des inconvénients. Nous pouvons classifier ces protocoles en quatre grandes familles. La première regroupe les protocoles à large bande passante et à courte portée, comme le Wifi qui est énergivore. La seconde famille prend en compte tous les protocoles à large bande passante et à longue portée comme la 3G ou la 4G. Les protocoles à faible bande passante et à faible portée comme le Bluetooth ou le ZWave sont regroupés au sein de la troisième famille. Enfin, par déduction, la qua-

trième famille regroupe les protocoles à faible bande passante sur une longue distance comme c'est le cas pour SigFox ou LoRa. Cette catégorie connaît un fort développement actuellement, notamment car ces protocoles consomment très peu d'énergie et permettent de concevoir des objets connectés avec une durée de vie très importante (10 années ou plus).

Pour connecter tous ces périphériques à l'Internet, l'on utilise généralement des passerelles (gateways) qui permettent d'interconnecter différents réseaux travaillant avec des protocoles différents. Ces passerelles garantissent la transparence avec les protocoles utilisés. Néanmoins, dû à cette transparence, aucun raisonnement n'est fait sur les spécificités des différents protocoles. Actuellement, il n'existe pas de mécanismes capables de raisonner sur les contraintes du matériel et des protocoles. Il est possible d'interconnecter des réseaux hétérogènes mais sans intégrer ces spécificités.

1.2.2.2 Augmentation des échanges de données

En plus de l'augmentation du nombre d'objets connectés, la quantité de données échangées est elle aussi en forte hausse. Il est possible de remarquer des changements sociétaux vis à vis de l'utilisation des données. Plutôt que de faire une acquisition de données lorsqu'une application ou un utilisateur en a besoin, la tendance actuelle est plutôt au stockage massif de toutes les données possibles en vue de les utiliser plus tard pour fournir un nouveau service après-coup.

C'est par exemple ce qui est prôné par la mouvance du « quantified-self » [Wolf 10, Swan 13]. Dans cette mouvance, les personnes impliquées relèvent toute une série de données personnelles de leur vie de tous les jours et les stockent dans le but de surveiller et d'améliorer leur style de vie. À la base, ces données étaient collectées par l'Homme sur papier. Mais avec l'avènement des objets connectés, les données sont maintenant collectées automatiquement, stockées sur des serveurs distants et des algorithmes de traitement de données peuvent maintenant extraire des informations pertinentes et fournir à l'utilisateur des conseils de manière automatique.

La tendance actuelle est donc à la supervision en temps réel de tout type de données. Ce qui pose des questions comme la saturation des réseaux de communication, ou la puissance de traitement de ces données.

1.2.2.3 Techniques pour le traitement massif des données

Afin de prendre en charge cette augmentation des échanges de données, les techniques de traitement ont évolué. Cela a débuté avec le calcul distribué, dans le but de traiter de gros volumes de données de manière efficace. Actuellement, les techniques de MapReduce [Dean 04] ont pris le dessus. Ce patron de programmation a été développé dans les laboratoires de Google pour traiter de gros volumes de données en utilisant un algorithme distribué de calcul parallèle sur des clusters de machines. Chaque nœud du cluster utilise une fonction Map qui a pour but d'appliquer une opération basique et séquentielle sur un ensemble de données. Le résultat en sortie est composé d'une

donnée sous la forme clé/valeur. Ces données sont redistribuées à d'autres nœuds en charge d'appliquer une étape Reduce. Ici, les données sont traitées par clé et la sortie produite représente un ensemble de données partielles et synthétisées qui sont ensuite collectées afin de produire le résultat final.

Divers frameworks ont été construits autour de cette technique de MapReduce. C'est le cas du framework open source Hadoop créé en 2009 par Doug Cutting, développé et maintenu par la fondation Apache, et utilisé par plusieurs organisations telles que Facebook, Amazon ou Yahoo. Ce framework aide à concevoir des applications orientées « batch » pour traiter des petaoctets de données réparties en lots, de manière hors ligne [White 12]. Néanmoins, les applications actuelles réclament de plus en plus des traitements en temps réel. C'est pourquoi d'autres frameworks, eux aussi basés sur la technique du MapReduce, ont vu le jour. Nathan Marz par exemple a développé Storm pour les besoins de Twitter. C'est un framework lui aussi open source qui permet de traiter des flux de données en temps réel sur cluster de serveurs homogènes et à l'aide d'un modèle de traitement des événements complexes (CEP [2]) [Nathan 11].

L'évolution du traitement des données n'est bien entendu pas centrée uniquement sur la technique du MapReduce. Les frameworks tels que Hadoop et Storm fournissent également toute une série d'outils qui facilitent la manipulation des données, la gestion de leur partage, leur synchronisation, leur stockage et leur mise à disposition. Des outils comme Zookeeper de la fondation Apache fournit un service centralisé pour la synchronisation décentralisée et pour le maintient de la consistance de données de configuration sur de larges systèmes distribués. Des systèmes de fichiers distribués comme HDFS ont aussi été développés pour permettre la transparence pour le stockage de fichiers au sein de clusters. Pour stocker des données, les bases de données ont aussi évolué. D'un modèle classique de bases de données relationnelles orientés lignes, nous sommes passés à des modèles non uniquement relationnels (NoSQL), orientés colonnes, comme Cassandra, HBase et maintenant, BigTable de Google, utilisés par Facebook, Twitter, Amazon ou Google afin de gérer efficacement des petaoctets de données. Se développent aussi des bases de données orientées graphes comme OrientDb ou Neo4j, utilisées par eBay ou Meetic. Ce nouveau type de base de données NoSQL est particulièrement efficace pour stocker des données qui sont aptes à être modifiées fréquemment et qui n'ont pas de schéma relationnel fixe. De plus, la représentation graphe permet aussi de faire de puissantes requêtes à l'aide de langages de requêtes spécifiques comme Gremlin, qui vient exploiter les propriétés des graphes.

Toutes ces techniques, frameworks et outils ont été conçus afin de permettre un traitement efficace de données à l'aide d'ensembles plutôt homogènes d'ordinateurs. Ce traitement est exécuté de manière transparente pour l'utilisateur. Ce dernier conçoit son procédé, ajoute des données en entrée et obtient le résultat en sortie, sans se soucier de l'endroit où est exécuté le traitement, ni du nombre de machines impliquées. Les spécificités des composants matériels peuvent être ignorées dans le cas de ces environnements relativement homogènes, mais elles doivent être prises en compte si l'on souhaite déployer des applications distribuées au sein de systèmes ambiants.

2. Complex Event Processing

1.2.3 Le problème de la protection de la vie privée

1.2.3.1 Les mauvais côtés des techniques actuelles

Nous venons de voir dans les sections précédentes que le développement de l'Internet des Objets permet de connecter tout type de périphériques afin de générer, traiter et transmettre de gros volumes de données au travers de l'Internet. Ces innovations et améliorations permettent d'imaginer de nouveaux champs d'applications et de proposer des innovations. Néanmoins, le caractère nouveau de ces techniques entraîne un manque de maturité qui empêche d'établir un ensemble de bonnes pratiques.

Il n'existe pas encore de mécanismes capables de faciliter l'interconnexion entre deux périphériques utilisant des technologies différentes. A cause de cela, les entreprises commercialisant des objets connectés prônent une connexion verticale entre le périphérique et les serveurs de l'entreprise. Les business models ont été conçus en se basant sur ce principe, ce qui conduit à la production d'objets fermés. Les données sont directement récupérées par les entreprises qui possèdent donc nos données et les utilisent soit pour fournir aux utilisateurs des services spécifiques, soit pour les revendre à des tierces parties. Dans tous les cas, l'utilisateur a accès aux services et non aux données elles-mêmes. Cette approche conduit inévitablement à un effet Big Brother : l'utilisateur ne possède plus ses données.

De plus, le Cloud computing ainsi que les techniques de Big Data fournissent des mécanismes facilitant énormément la synchronisation, le traitement et le partage des données. Néanmoins, ces techniques transportent les données au travers de l'Internet et demandent énormément de bande passante. Dans certains cas, c'est indispensable, notamment pour les applications de réseaux sociaux, comme Twitter, qui doivent traiter les données à une échelle globale. Mais ce n'est pourtant pas le cas de la plupart des applications. Celles permettant de surveiller ses activités journalières ou de gérer un bâtiment produisent des données qui peuvent être traitées localement. Dans ces cas, il n'y a pas besoin d'exporter ces données au travers du monde. Il nous faut trouver des mécanismes « multi-scale » et définir des politiques de sécurisation des données afin de permettre l'exportation de certaines données seulement, potentiellement agrégées, dans le Cloud.

1.2.3.2 La vie privée : un phénomène de société

Toutes ces technologies ont été imaginées pour faciliter la conception d'applications intelligentes pour l'utilisateur. Néanmoins, il faut avant tout que l'utilisateur accepte d'utiliser ces applications et adoptent ces technologies. Ceci ne sera rendu possible que lorsque que l'on garantira à ces utilisateurs la confidentialité de leurs données et de leurs ressources [Zhou 10, Kirchbuchner 15].

Ainsi, en 2007 lors de la conférence IJCAI [3], Chopra et White [Chopra 07] publiaient un article au titre provocateur : « Privacy and Artificial Agents, or, Is Google Reading My Email ». Dans celui-ci, les auteurs proposent une analyse philosophique et légale

3. the International Joint Conference on Artificial Intelligence

d'un certain nombre de problèmes liés à la sécurité des informations des usagers. Un exemple marquant de problème lié à la confidentialité est celui de savoir si Google lit les mails de ses utilisateurs ou non. Google répond que la confidentialité n'est pas remis en cause car aucun humain autre que l'utilisateur ne peut lire ses emails. Néanmoins, Google analyse le contenu des messages afin de filtrer le spam, détecter les virus, mais aussi identifier une partie du texte qui peut servir à proposer aux utilisateurs un ensemble de publicités personnalisées ! Une des réponses faite à Google sur ce point est qu'un système informatique doté d'une plus grande capacité de stockage et d'analyse qu'un humain, peut être aussi invasif, si ce n'est plus.

> *« ... a computer system, with its greater storage, memory, and associative ability than a human's, could be just as invasive as a human listening to the communications, if not more so. »*

Un certain nombre d'incidents et de controverses au sujet de la confidentialité des données ont d'ailleurs déjà été recensés parmi les produits du Cloud computing ou de l'Internet des Objets vendus ou mis à disposition du public. Ainsi, par exemple, en 2009 un problème a été découvert dans les Google Docs qui a causé le partage intempestif de documents Google sans que les autorisations ne soient validées. Chez Microsoft, la XBox One qui activait constamment le microphone afin de réagir à certaines phrases prononcées par l'utilisateur. C'est aussi le cas pour le mode de recherche Google sur smartphone qui réagit à la phrase "Ok Google Now" afin de lancer une recherche. Ce sont deux exemples d'applications potentiellement dangereuses pour le respect de la vie privée : ces systèmes peuvent récupérer l'ensemble des données vocales et envoyer ces informations à Google ou à Microsoft dans le but de détecter de potentielles phrases compatibles. Nous ne pouvons néanmoins pas savoir si ces données sont utilisées à d'autres fins.

La question n'est pourtant pas de savoir si ce type de systèmes exploite ou non toutes ces données sans l'autorisation de l'utilisateur. Ce qui est intéressant, c'est de noter que l'utilisateur s'inquiète de ce type d'intrusion et que rien actuellement ne peut le rassurer en ce qui concerne la confidentialité des données. Pire que ça, à cause du manque de mécanismes permettant de garantir cette confidentialité, les utilisateurs sont de plus en plus réticents à utiliser ces technologies. Même si beaucoup de monde diffuse ses données personnelles sur des sites tels que Facebook, ou sur d'autres applications sans se préoccuper de leur confidentialité et sans réaliser que ces informations ne leur appartiennent plus, certaines autres personnes, a contrario, militent contre de tels abus ou, dans les cas extrêmes, tombent dans la paranoïa. Afin d'éviter ce malaise et mettre en avant l'utilisation d'applications ambiantes, il faut donc proposer des systèmes qui garantissent la confidentialité des données et des ressources aux utilisateurs.

1.3 Conclusion

Nous avons débuté ce chapitre en présentant un monde dans lequel les objets électroniques qui nous entourent sont connectés à un réseau interopérable et peuvent communiquer afin de fournir de manière totalement transparente à l'utilisateur un ensemble

de services correspondant à ses besoins. Bien évidement, la réalisation de cette vision du futur soulève un grand nombre de problèmes que les solutions proposées actuellement dans l'industrie n'ont pas encore réussies à surmonter. Parmi ces problématiques, nous avons retenu :

— L'hétérogénéité du matériel et des protocoles de communication qui empêche la conception d'applications génériques et leurs interactions.
— Indirectement ce problème entraîne la question du transport des données : pour pouvoir faire coopérer ces objets, les données sont souvent centralisées et traitées loin du capteur, saturant du même coup la bande passante des réseaux de communication.
— Enfin, la question de la confidentialité est aussi soulevée ; les données des objets étant envoyées sur des serveurs externes au travers des réseaux de communication, il est indispensable de fournir des mécanismes garantissant la confidentialité de celles-ci.

Nous dresserons au chapitre suivant un état de l'art des travaux de recherche qui répondent à ces problématiques.

Chapitre 2

État de l'art

Sommaire

Dans le chapitre précédent, nous avons isolé un certain nombre de problématiques freinant ou guidant dans les choix de conception des solutions pour la gestion d'environnements intelligents. L'hétérogénéité du matériel et des protocoles, l'échange massif au niveau global de données, ainsi que la confidentialité de ces données ont pourtant été considéré dans de nombreux travaux. Ce chapitre présente un état de l'art de ces différents travaux de recherche permettant de prendre en compte ces problématiques.

La Section 2.1 dresse un état de l'art des différentes applications et intergiciels existants pour la gestion des environnements intelligents. Ces applications et intergiciels essayent à leur manière de répondre aux problèmes de l'hétérogénéité du matériel et des protocoles ainsi qu'au problème de la confidentialité des données et des informations de l'utilisateur. La Section 2.2 fait l'état des lieux des travaux sur le déploiement automatique d'applications sur des systèmes distribués considérant l'hétérogénéité de l'infrastructure ainsi que le besoin de pouvoir passer à l'échelle. Enfin, la représentation des connaissances étant un problème commun à la majorité des travaux qui seront présentés dans ce chapitre, la Section 2.3 présente les différents formalismes permettant de représenter et de raisonner sur les connaissances. Cette dernière section permettra de comprendre plus facilement les choix qui ont été faits concernant la manière dont nous modéliserons les environnements ambiants (Chapitre 3) ainsi que le choix de la méthode de raisonnement pour déployer des applications (Chapitres 4 et 5).

2.1 Gestion des environnements ambiants

2.1.1 Applications pour la gestion d'environnements

Beaucoup de travaux en Intelligence Ambiante se focalisent sur l'implémentation de cas particuliers. Certains auteurs font un état de l'art des applications de gestion des environnements ambiants dans différents domaines d'applications. Ramos et Al. présentent quantités de projets de gestion de bureaux intelligents et de maisons intelligentes [Ramos 10]. Kameas et Al. recensent les principaux projets de systèmes pervasifs appliqués à la santé [Kameas 10] Enfin, Bohlen et Al. fait un état de l'art sur la gestion de villes intelligentes [Böhlen 10]. Nous présentons dans cette sous-section quelques une de ces applications.

Un des premiers travaux prometteur concernant la gestion d'un environnement intelligent de type maison est celui de Mozer et Al. [Mozer 98, Mozer 04]. Les auteurs présentent ACHE, un environnement capable de s'adapter aux habitants d'un logement grâce à un réseau de neurones qui permet de récupérer les valeurs des capteurs de la maison et de contrôler les différents actionneurs pour gérer des opérations classiques en domotique : le contrôle des lumières, de la température et de la ventilation. Néanmoins, cette proposition, comme la majorité des travaux des années 1980 et 1990 en domotique, repose sur un système centralisé et statique qui ne prend en compte ni la dynamicité de l'environnement, ni son hétérogénéité puisque la solution proposée se concentre sur la méthode d'inférence et non sur les questions relatives à la récupération des informations des différents capteurs.

Hagras et Al. proposent iDorm [Hagras 04], un prototype pour la gestion de lieux de vie (bureau, maison, chambre d'hôtel, etc.). Dans ce projet, les capteurs et les actionneurs de l'environnement (lumière, température, etc.) sont connectés au réseau et remontent des informations permettant à un algorithme d'apprentissage à base de logique floue afin de prédire les besoins des utilisateurs et s'adapter en fonction des changements dans l'environnement. Les auteurs proposent trois types d'éléments (artifacts). Le premier type est un agent statique qui prend en charge le bâtiment en récupérant via le réseau les informations des capteurs et traitent ces données à l'aide de l'algorithme à base de logique floue. Le second type représente des agents robotiques capables de se déplacer dans l'environnement en se coordonnant à l'agent statique pour adapter automatiquement sa navigation. Le dernier type correspond aux artifacts de calcul embarqués qui permettent de fournir une hiérarchie de passerelles récupérant les informations des capteurs et les relayant à l'agent statique. L'apprentissage et la coordination se font de manière non intrusive sans l'aide de l'utilisateur. Ce dernier intervient uniquement lorsque le système effectue une action qui ne lui plaît pas. Ce projet met toujours en œuvre un système centralisé autour de l'agent statique, mais introduit le concept de passerelles permettant de relier les capteurs au réseau et de relayer ses informations à d'autres entités en charge du raisonnement.

EasyMeeting est un système à base d'agents pour la gestion de salles de réunion. Proposé par Chen et Al. [Chen 04], ce projet a pour particularité de proposer, outre un système multi-agents encapsulant chacune des capacités des entités matérielles de l'environnement, une ontologie permettant de décrire les informations contextuelles liées à la salle de réunion. Ces informations regroupent le profil de l'utilisateur, la planification des réunions ainsi que des informations temporelles et spatiales. Cette ontologie fournit donc un cadre commun aux différents agents leur permettant à la fois d'interagir entre eux, mais aussi de raisonner sur la planification des différentes actions à effectuer à l'aide de systèmes à base de règles d'hypothèses et de détecter les inconsistances. Un système de politiques de confidentialité permet aux utilisateurs de contrôler le partage et l'utilisation de leurs informations contextuelles. La Figure 2.1 représente l'architecture du gestionnaire de contexte de ce projet.

MyCampus est un projet de gestion d'environnements ambiants présenté par Sadeh et Al. [Sadeh 05]. Ce projet met en œuvre une ontologie qui permet de décrire des services contextuels en prenant en compte certains aspects de la confidentialité de ces informations contextuelles. Chaque source d'information contextuelle est représentée par un service web (Semantic Web service). La solution est architecturée autour d'agents appelés « semantic e-Wallets » qui regroupent l'ensemble des ressources contextuelles de chaque utilisateur. Cet agent sauvegarde les préférences de confidentialité de son utilisateur afin de contrôler les informations qu'il peut partager.

DALICA est un système multi-agents créé par Costantini et Al. [Costantini 08] pour la gestion de la dissémination des informations pour les sites culturels. Un agent assistant guide les visiteurs au travers d'un site culturel en leur suggérant un itinéraire en fonction de leurs intérêts. Malheureusement, cette solution est spécifique aux sites culturels et ne peut pas être étendue à d'autres applications plus génériques.

Les exemples d'applications présentées dans cette sous-section montrent le besoin d'adopter une architecture décentralisée pour gérer les entités matérielles de manière

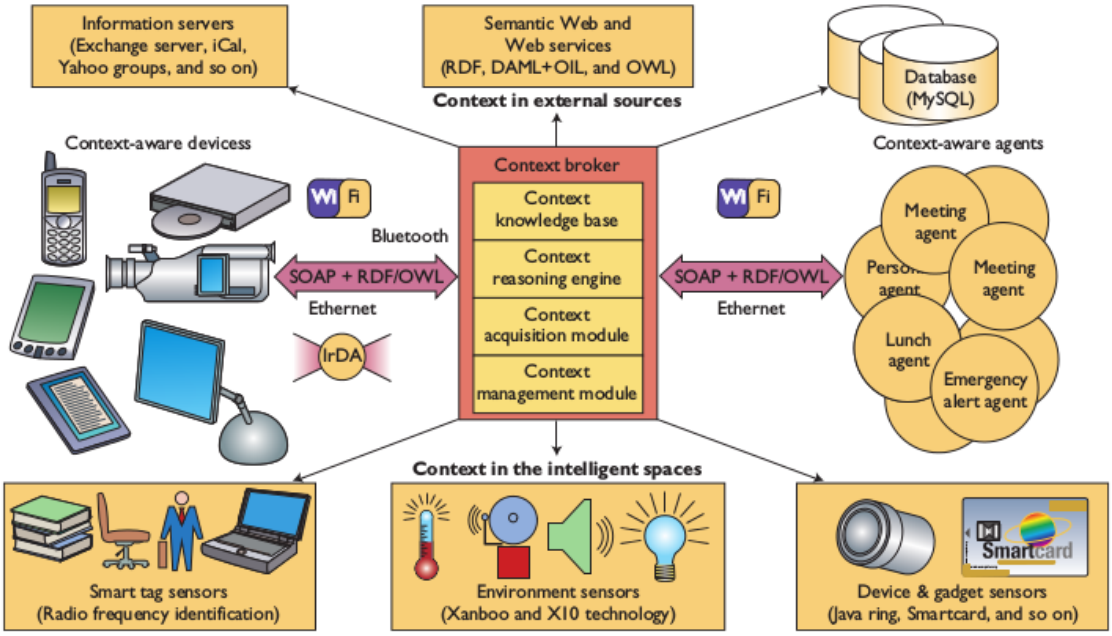


FIGURE 2.1 – Architecture du gestionnaire de contexte de la plateforme easyMeeting

dynamique. Dans ces projets, la gestion de cette hétérogénéité passe souvent par le développement d'ontologie permettant d'avoir un cadre commun pour exprimer le type d'informations à prendre en compte et ainsi de permettre aux différentes entités du système distribué de communiquer. Certains autres travaux se proposent de définir un cadre générique sous la forme d'intergiciel pour la gestion des environnements ambiants et la conception d'applications en son sein.

2.1.2 Intergiciels pour la gestion des environnements ambiants

A cause de l'hétérogénéité importante des systèmes ambiants et pour permettre une implémentation générique pouvant passer à l'échelle, il est nécessaire de passer par la création d'intergiciels offrant des fonctions de base pour la gestion d'environnements intelligents [Schiele 10]. La plupart des travaux sur les intergiciels pour l'Intelligence Ambiante se focalisent sur la gestion du contexte et proposent des ensembles d'outils et de mécanismes afin de collecter, raisonner et fournir des informations relatives au contexte lié à l'utilisateur et à l'environnement. Certains travaux dressent un état de l'art de différents intergiciels pour les environnements intelligents [Fortino 14].

La plateforme SpatialAgents proposé par Satoh et Al. [Satoh 04] met en œuvre des agents mobiles offrant des fonctionnalités aux utilisateurs. L'architecture de cette plateforme, illustrée à la Figure 2.2, possède de trois composants : un système de localisation, un fournisseur de services et des agents mobiles. Un système de localisation vient identifier et localiser les périphériques présents dans un espace. Un fournisseur de services délivre des services correspondant aux capacités et aux fonctionnalités des différents périphériques et espaces. Chacune de ces fonctions est encapsulée dans un agent mobile indépendamment de n'importe quelle application. Les utilisateurs sont

représentés dans le système au travers d'agents personnels gardant à jour les préférences de ceux-ci. Lorsqu'un périphérique rentre dans un espace, il est identifié grâce au système de localisation. Des agents encapsulant les fonctionnalités du périphérique sont alors déployés sur certaines machines contenant des « Agent host » permettant d'exécuter des agents Java. Ces agents offrent aux utilisateurs présents dans l'espace les services associés. Lorsqu'un périphérique sort d'une zone, l'agent associé migre automatiquement vers le fournisseur de service. Chaque agent est implémenté en Java et possède une interface graphique s'adaptant au type d'écran [Satoh 00]. Ce travail

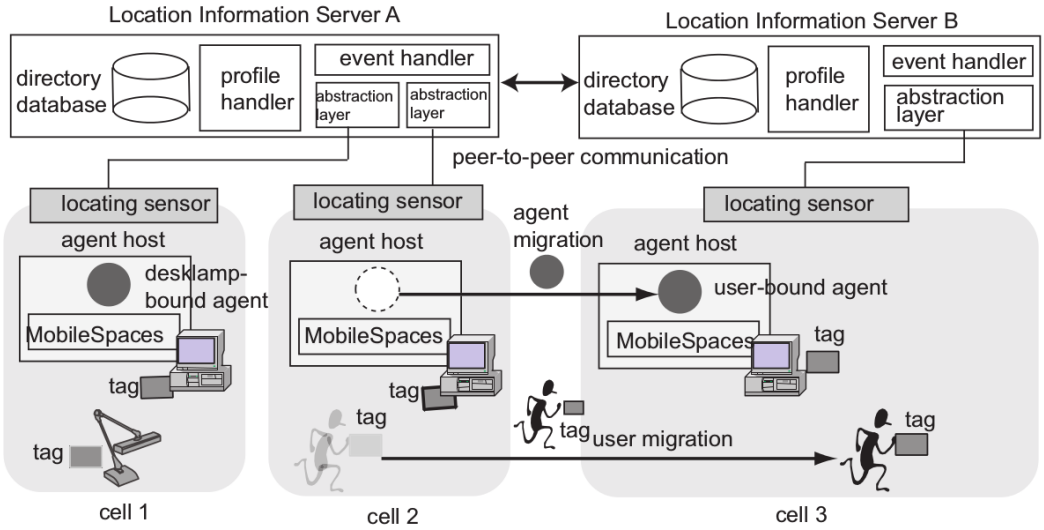


FIGURE 2.2 – Architecture de la plateforme SpatialAgent

prend en compte la dynamicité de l'environnement en permettant à de nouvelles entités matérielles d'entrer ou de sortir de l'environnement tout en exposant leurs capacités. Néanmoins, l'utilisateur ne peut faire qu'utiliser une entité matérielle au travers d'un service. Les entités ne peuvent pas communiquer directement entre elles ni entre les différentes pièces.

AmbieAgents est une plateforme agent proposée par Lech et Al. [Lech 05] qui se concentre sur l'acheminement d'informations aux utilisateurs mobiles en se basant sur le contexte. L'architecture proposée est composée de trois parties : (1) un ensemble de « tags contextuels » sans-fil ; (2) des utilisateurs mobiles ; et (3) des fournisseurs de services. Les « tags contextuels » sont un ensemble de micro-ordinateurs placés à des points stratégiques de l'environnement afin de relayer les informations des fournisseurs de services aux périphériques mobiles. Ces micro-ordinateurs possèdent des informations sur leur environnement et communiquent avec les périphériques mobiles par Bluetooth afin de récupérer la position de l'utilisateur ainsi que de fournir à ces derniers des informations contextuelles comme la liste des services disponibles par exemple. Les informations contextuelles prises en compte par cette plateforme sont des informations sociales, d'actions, personnelles, environnementales et spatio-temporelles [Myrhaug 01]. Cette classification de l'information constitue une ontologie qui permet aux agents de communiquer entre eux. Les agents, implémentés à l'aide de JADE [1] [Caire 03], sont regroupés en trois types. L'agent contextuel est le cœur de la plateforme puisqu'il per-

1. Java Agent Development framework

met d'accéder au contexte des utilisateurs. L'agent de contenu permet de fournir du contenu aux utilisateurs. Ces agents utilisent les informations anonymisées fournies par les agents contextuels afin de sélectionner le type de contenu à envoyer. Enfin, les agents de recommandation raisonnent sur l'état actuel de l'utilisateur afin de demander aux agents de contenu de fournir du contenu plus spécifique. Cette plateforme s'intéresse à la gestion du contexte et définit un modèle de représentation de ces informations contextuelles. Néanmoins, les travaux se placent au niveau de la couche de services et aucun mécanisme de gestion des entités matérielles n'est défini.

Le projet LAICA [2], introduit par Cabri et Al. [Cabri 05], a pour objectif de définir les modèles et les technologies ayant les bonnes propriétés pour la gestion d'environnements ambiants. Les auteurs apportent des arguments en faveur de l'utilisation du paradigme agent dans l'implémentation d'applications ambiantes. Ce projet propose un intergiciel agent pour l'Intelligence Ambiante. La Figure 2.3 représente l'architecture de celui-ci. Cet intergiciel propose deux types agents agents permettant d'interagir avec

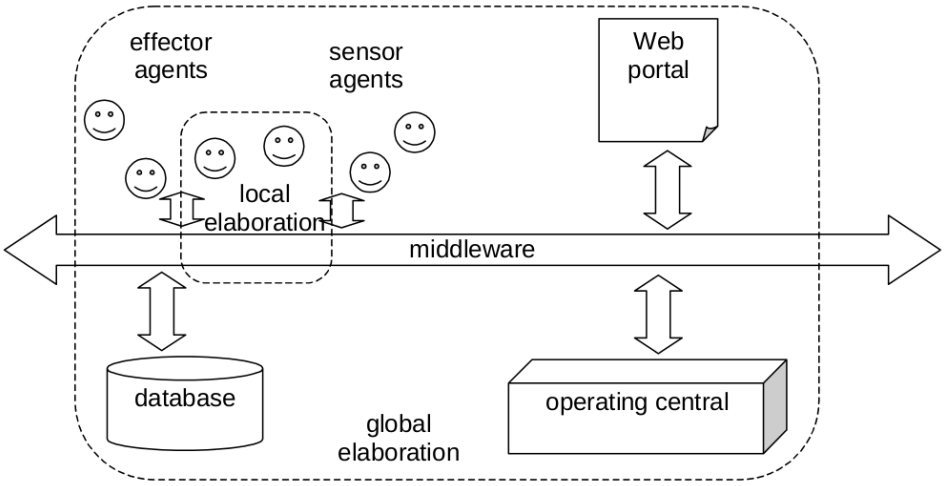


FIGURE 2.3 – Architecture du middleware LAICA

l'environnement. Les « sensor agents » sont chargés de la gestion des informations récupérées par les périphériques présents dans l'environnement. Les « effector agents » sont des agents capables d'effectuer une action dans l'environnement. Toutes les informations sont enregistrées dans une base de données et sont accessibles via un portail web. Enfin, la centrale d'opération est le cœur de l'architecture en charge de coordonner tous les autres composants. Malgré la simplicité du comportement des agents, leurs actions groupées (interaction, coordination, organisation) font que le comportement global du système est considéré comme intelligent. Bien que l'idée d'utiliser le paradigme agent pour gérer un environnement intelligent soit bien argumentée, l'intergiciel présenté dans ce travail n'utilise le paradigme agent que pour interagir avec l'environnement. Tous les autres entités de l'intergiciels ne sont pas agentifiées ni même décentralisées.

Seghrouchni et Al. [Seghrouchni 08] propose la plateforme CAMPUS dans l'optique de faciliter le développement d'applications multi-agents prenant en compte l'aspect contextuel pour les systèmes ambiants. Cette plateforme autorise l'utilisateur à sélec-

2. Laboratorio di Ambient Intelligence per una Città Amica

tionner un ensemble de services à activer dans son environnement. Elle est composée de trois niveaux : (1) la couche de gestion du contexte ; (2) la couche de communication et de coordination ; et (3) la couche des services ambiants. La couche de gestion du contexte a été implémentée en utilisant l'intergiciel orienté-service MoCA [3] [Sacramento 04]. Cette couche fournit des informations sur les entités matérielles, sur les services présents dans l'infrastructure ainsi que certaines fonctionnalités de base comme la découverte automatique des services. La couche de communication et de coordination est composée d'agents représentant les composants physiques de l'environnement, d'un protocole de communication et de coordination ainsi qu'une ontologie permettant de gérer les services, de les orchestrer et de les coordonner [Charif-Djebbar 06]. Les agents sont gérés (chargés, lancés et tués) par un service de gestion d'agents et un système de pages jaunes est proposé pour permettre aux agents de se découvrir mutuellement Enfin, la dernière couche fournit les applications et les services ambiants. Mis à part certains services comme l'ontologie ou le répertoire de services, l'architecture est cette fois-ci entièrement distribuée.

Olaru et Al. [Olaru 13] présente un intergiciel multi-agents pour l'Intelligence Ambiante pour le transfert d'informations contextuelles. Cet intergiciel doit répondre à quatre particularités :

— l'acheminement d'informations contextuelles de l'environnement vers l'application ou l'utilisateur concerné ;
— le transfert de nouvelles informations concernant l'utilisateur et ses activités vers les parties concernées ;
— la détection du contexte de l'utilisateur ;
— la décision sur la bonne action à effectuer.

Deux types d'entités sont présentées : les containers et les agents. Les containers sont liés à une entité de l'environnement et peuvent accueillir un certain nombre d'agents. Ces containers s'exécutent sur des machines de l'environnement. Les agents quant à eux prennent en charge les utilisateurs ainsi que certaines fonctionnalités des entités de l'environnement et communiquent et interagissent ensemble. Ils s'exécutent dans les containers et sont mobiles. Chaque agent possède une base de connaissance relative au contexte de l'entité prise en charge. Ces connaissances sont représentées sous la forme de graphes. L'intergiciel est focalisé sur son grand degré de distributivité ainsi que sur l'isolation des processus relatifs au transfert d'informations entre les périphériques. Cet intergiciel a été testé dans les projets Ao Dai [El Fallah-Seghrouchni 10] qui implémente un scénario de bâtiment intelligent dans lequel des agents prennent en charge : (1) les étages du bâtiment ; (2) les pièces ; (3) les entités matérielles, et en particulier le PDA [4] de l'utilisateur ainsi que les écrans du bâtiment ; (4) des services tels qu'un agenda et un plan du bâtiment. Ce scénario illustre la mobilité des agents : l'agent gérant le PDA de l'utilisateur migre d'étage en étage lorsque l'utilisateur se déplace ; les agents gérant les services sont instanciés puis envoyés sur le PDA de l'utilisateur. Un second projet utilisant cet intergiciel et mettant en œuvre une plateforme nommée tAIAmI [5] [Olaru 15] a été développé afin de tester des applications d'Intelligence Ambiante à base d'agents. Cet intergiciel reprend un bon nombre des points forts des travaux précédemment présentés : (1) une architecture décentralisée mettant

3. Mobile Collaboration Architecture
4. Personal Digital Assistant
5. toward Agent Technologies for Ambient Intelligence

en avant la coopération d'entités autonomes plutôt que le contrôle global ; (2) la gestion du contexte à base d'une ontologie définissant un cadre commun ; (3) la possibilité de migration des capacités des entités matérielles encapsulées dans les agents.

Comme les derniers travaux cités, d'autres travaux sont basés sur la proposition d'intergiciels à base d'ontologie pour la gestion d'application « context-aware ». C'est le cas des travaux de Babu et Al. [Babu 14, Babu 15] qui récupèrent les données de capteurs physiques ou virtuels dans des fichiers XML, interprètent et fusionnent ces données afin de générer des informations contextuelles qui seront stockées dans une base de données pour une utilisation future. Un système d'inférence à base de règles permet d'analyser le contexte courant pour en déduire un ensemble de conclusions qui lui permet de s'améliorer. Les travaux de Gómez-Goiri et Al. [Gómez-Goiri 14] ou encore de Stavropoulos et Al. [Stavropoulos 12] proposent aussi un intergiciel basé sur une ontologie pour rendre les applications ambiantes interopérables.

Cette recherche d'interopérabilité se retrouve dans des travaux sur des intergiciels à base de services dans lesquels la recherche de l'interopérabilité entre les applications et les entités matérielles a remplacé le besoin de fournir un environnement « context-aware ». C'est le cas par exemple de l'intergiciel Amigo développé par Georgantas et Al. [Georgantas 10]. Amigo utilise une architecture orientée services qui permet de faciliter l'interopérabilité. L'originalité de cette approche comparée à tous les travaux présentés jusqu'à maintenant est que cet intergiciel n'impose pas de technologie particulière, mais autorise au contraire l'intégration de technologies hétérogènes. L'interopérabilité est effectuée à un niveau sémantique plus élevé utilisant des ontologies écrites en OWL [6] permettant de décrire un certain nombre de connaissances relatives aux capacités fournies par les services. D'autres intergiciels sont basés sur le même principe d'une infrastructure orientée services qui intègre et encapsule les entités matérielles hétérogènes de l'environnement dans des services. C'est le cas par exemple de l'intergiciel AWESoME à base de services Web [Stavropoulos 13], ou encore du projet Hydra [Eisenhauer 10].

2.1.3 Confidentialité dans les systèmes distribués

La sécurité dans les système distribués et dans les systèmes informatiques en général est un point clef du domaine des systèmes d'informations. Avec la complexification des systèmes (ordinateurs organisés en réseaux devenus pervasifs), un certain nombre de problèmes et d'interactions inattendus provoquent inévitablement des comportements non désirés des systèmes. Ces nombreux problèmes peuvent être exploités par des utilisateurs malintentionnés dans le but d'accéder ou d'agir sur des ressources qui ne leur appartiennent pas (récupération d'un mot de passe, suppression intempestive de données, etc.). La sécurité en ingénierie logicielle peut être définie comme étant la manière de gérer ces risques [Stajano 10]. Ces risques peuvent être perçus de deux points de vue différents :

— Le point de vue amont lors de la conception de système. On essaye dans ce cas d'imaginer les différents problèmes qui peuvent potentiellement intervenir dans un système, puis l'on met en place des architectures (logicielles, réseaux,

6. Web Ontology Language

infrastructurelles) robustes permettant de prévenir ces problèmes.
— Le point de vue aval lorsque des comportements étranges surviennent dans un système déjà en production. Dans ce cas, le domaine de la sécurité vise à identifier les problèmes et leurs sources afin de proposer des solutions adhoc pour les éradiquer.

Traditionnellement, les différents risques que l'on retrouve dans les systèmes d'informations peuvent affecter trois des propriétés fondamentales que doit respecter un système :
— la confidentialité des données/ressources : seules les personnes autorisées ont le droit de consulter un certain groupe de données ou d'accéder à un certain nombre de ressources ;
— l'intégrité des données/ressources : seuls les propriétaires des données ou des ressources ont le droit de les modifier ou de les utiliser ;
— l'accessibilité des données/ressources : les personnes autorisées à accéder à des données ou des ressources doivent pouvoir le faire instantanément.

C'est le premier point concernant la confidentialité des données et des ressources qui nous intéresse dans le cadre de cette thèse. La confidentialité est définie par Westin [Westin 67] comme étant le droit aux individus, aux groupes ou aux institutions, de déterminer pour eux-même, quand, comment et quelles informations à propos d'eux peuvent être partagées. La confidentialité dans les systèmes distribués a déjà été bien explorée [Mabrouki 14]. Solove et al. [Solove 06] ont catégorisé les recherches sur la confidentialité des données en différents niveaux : la collecte, le traitement, la divulgation et la diffusion. La collecte d'informations fait référence au processus de récupération et de stockage des données d'un individu ou à propos d'un individu. Le traitement des informations fait référence à la manière d'utiliser et de transformer les données collectées. La divulgation des données fait référence à la manière dont une donnée peut-être accessible à un certain groupe de personnes. Enfin, la diffusion des informations fait référence au transfert des données collectées et/ou traitées à un parti tiers.

Nous avons vu dans les sous-sections précédentes que beaucoup de travaux utilisent les systèmes multi-agents afin d'encapsuler les données personnelles des utilisateurs au sein d'agents [Fasli 07]. De part leurs propriétés d'autonomie, de proactivité ou par leur aspect distribué, le paradigme multi-agents est adapté au développement de la confidentialité des systèmes distribués. Des systèmes multi-agents spécifiques ont donc été proposés afin de prendre en charge de différentes manières les aspects de la confidentialité de données listés ci-dessus [Such 14]. C'est le cas par exemple des travaux de Chen et Al. [Chen 04] ainsi que ceux de Sadeh et Al. [Sadeh 05] déjà présentés.

2.1.3.1 Politiques de confidentialité

Tentori et Al. [Tentori 06] proposent une extension du framework agent SALSA [Rodríguez 08] permettant la gestion de la confidentialité de informations échangées entre les agents. Chaque agent possède deux politiques de confidentialité, la première portant sur les informations que l'agent peut communiquer aux autres agents et la seconde concernant les informations dont l'agent a besoin. Ces deux politiques sont exprimées par le développeur à l'aide d'un schéma XML. Les informations transitent ensuite par un « Agent Broker » qui vérifie que les politiques des agents soient compa-

tibles et respectées. Les performances du système repose donc sur cette dernière entité qui constitue un unique point de défaillance potentielle (Single Point of Failure).

Crépin et Al. [Crépin 09] proposent une ontologie ainsi qu'un protocole d'interaction entre agents afin de préserver la confidentialité des données. L'ontologie, décrite en OWL [7], fournit les différents concepts permettant de définir les politiques de confidentialité des différents agents (temps de détention d'une donnée, utilisation possible, etc.). Comme dans les travaux de Tentori et Al., il existe deux politiques de confidentialité. La première pour les agents fournissant des données et la seconde pour les agents les consommant. Le protocole permet à un consommateur d'envoyer une requête afin d'obtenir certaines données d'un producteur. La politique de confidentialité est envoyée avec la requête et le producteur vérifie que celle-ci corresponde bien à sa propre politique de partage. Si c'est le cas, les données lui sont envoyées en réponse, sinon, le consommateur demande à l'agent l'envoi de sa politique. Ce travail ne prend pas en compte le fait qu'un agent consommateur peut potentiellement ne pas appliquer sa politique de confidentialité une fois qu'il a obtenu les informations qu'il souhaitait.

Udupi et Al. [Udupi 10] proposent une plateforme appelée InterPol supportant des politiques de confidentialité, ainsi qu'un langage. Ces politiques sont écrites à l'aide de règles Prolog et peuvent s'adapter dynamiquement aux changements de relation entre les agents en ajoutant de nouveaux prédicats à la base de connaissances de l'agent.

2.1.3.2 Normes et relations sociales

Barth et Al. [Barth 06] proposent une plateforme permettant d'exprimer et de raisonner sur des normes qui traitent de la transmission d'informations personnelles. Ces normes sont divisées en deux types : les normes positives et les normes négatives. Elles sont exprimées à base de logique temporelle et permettent de définir les permissions et les interdictions de partage d'informations privées concernant les agents du système. Une norme positive autorise la communication d'une information si sa condition temporelle est respectée. Au contraire, une norme négative permet la communication d'une information seulement si sa condition temporelle est satisfaite. Les auteurs considèrent que le système est fermé et que tous les agents respectent ces normes. Aucun mécanisme n'est prévu pour intégrer des agents externes qui ne joueraient potentiellement pas le jeu.

Pour contrer ce dernier problème, les travaux de Krupa et Al. [Krupa 10] considèrent un système multi-agents ouvert dans lequel les agents peuvent potentiellement ne pas respecter les normes. Ils proposent d'utiliser des modèles sociaux de réputation et de confiance afin de pénaliser les agents déviants. Les auteurs proposent cinq normes générales qui permettent de préserver la confidentialité lors de la dissémination d'une information entre plusieurs agents :

— les agents ne doivent pas transférer des informations inadaptées au contexte ;
— les agents doivent transmettre des informations relatives au parcours de l'information (par quel agent l'information est-elle passée) ;
— ne pas envoyer d'information à des agents connus pour violer les normes établies ;

7. Web Ontology Language

— ne pas tenir compte des informations reçues par des agents connus pour violer les normes établies ;
— envoyer aux agents violant les normes des messages les informant qu'ils ont enfreint les règles établies.

De nombreux modèles de relations sociales basées sur la confiance, l'intimité ou la réputation ont été développés au sein de la communauté agent [Ramchurn 04, Sabater 05]. C'est en utilisant ces travaux, ainsi que grâce aux préconisations de Krupa et Al. que Such et Al. proposent d'utiliser les relations sociales pour sélectionner les agents avec lesquels les données peuvent être partagées [Such 12]. Les auteurs proposent un modèle de confiance qui prend en compte la divulgation d'un agent par rapport aux autres agents ainsi que la divulgation d'informations de ces autres agents en retour. Le modèle prend aussi en compte la fiabilité d'un agent en vérifiant que les informations révélées sur lui-même sont justes. Malheureusement, certaines informations ne peuvent pas être vérifiées.

2.1.3.3 Partis tiers

Aïmer et Al. [Aïmeur 06] proposent ALAMBIC, une architecture mettant en œuvre des utilisateurs, un fournisseur de services et une entité appelée « Still Maker ». Cette entité est une plateforme sécurisée générant des agents mobiles qui servent d'entité tierce. Ces agents migrent vers le fournisseur de services et ont pour mission de filtrer les informations concernant l'utilisateur à l'aide de techniques de filtrage démographique. Ces données utilisateurs sont encryptées et seules sont divulguées les informations que l'agent considère pouvoir divulguer. Cissée et Al. étendent ces travaux en proposant de faire migrer l'agent non plus chez le fournisseur de services, mais sur une plateforme indépendante et sécurisée [Cissée 07].

Menczer et Al. présentent IntelliShopper, un assistant intelligent qui aide et conseille l'utilisateur en lui proposant des produits sur des sites d'e-shopping qui pourraient l'intéresser [Menczer 02]. Pour cela, l'agent récupère des informations concernant les activités et les préférences de l'utilisateur afin de déterminer les produits intéressants à surveiller. Les utilisateurs sont connectés à IntelliShopper au travers d'une entité anonymisant celui ci. Ainsi, la plateforme ne possède pas d'informations permettant d'identifier l'utilisateur (par exemple, son adresse IP). Le même principe d'entité tierce permettant d'anonymiser les utilisateurs est repris dans différents travaux comme ceux de Jaiswal et Al. qui proposent l'architecture MAGNET utilisant des « anonymisateurs » entre une place de marché et des enchérisseurs [Jaiswal 05].

2.1.3.4 Sécurisation des communications

Enfin, des travaux [Alberola 10] se focalisent sur l'intégration de moyens de communication sécurisés dans les plateformes orientées agents en utilisant des protocoles de chiffrage bien connus.

Tous les travaux cités dans cette Sous-Section 2.1.3 utilisent des SMA afin d'améliorer la confidentialité des données et des informations de l'utilisateur. Chopra et Al.

[Chopra 07] encouragent à intégrer une réflexion sur la confidentialité des données dès la phase de conception d'un système multi-agents. Dans nos travaux, comme nous le verrons dans les chapitres suivants, nous utiliserons les avantages des propriétés des SMA pour prendre en charge uniquement la confidentialité des ressources et de la structure de notre système.

2.1.4 Synthèse

Dans cette section, nous avons présenté un certain nombre de travaux relatifs à la gestion des environnements intelligents d'un point de vue applicatif, intergiciel et de la confidentialité. La plupart des applications sont conçues autour de certains principes :

— la conception d'architectures distribuées à base d'agents et/ou de services ;
— l'encapsulation des entités matérielles ou de leurs capacités au sein d'agents ou de services ;
— le développement d'ontologies pour l'interopérabilité ou la gestion du contexte.

Les intergiciels quant à eux se focalisent principalement sur l'abstraction de mécanismes permettant la prise en compte du contexte dans les applications ambiantes. Ces intergiciels sont conçus pour faciliter le développement d'applications distribuées « context-awares ». Enfin, les travaux sur la confidentialité des informations dans les systèmes distribués proposent un certain nombre de techniques basées sur les politiques de confidentialité, les normes, les relations sociales ainsi que sur la prise en charge des données par des partis tiers de confiance.

Tous ces travaux et en particulier ceux sur les intergiciels facilitent le développement d'applications pour des environnements intelligents en fournissant des outils utilisables par les développeurs. La plupart des intergiciels imposent donc un certain nombre de technologies. Par ailleurs, aucun travail ne se place à un niveau d'abstraction plus important, laissant le développeur choisir entièrement la manière de développer ses applications, en se concentrant uniquement sur les mécanismes permettant de raisonner sur le choix des entités matérielles à utiliser pour faire tourner ces applications. Pourtant, le déploiement automatique d'applications est un domaine très actif depuis les années 1990. Les différentes technologies proposées pourraient être appliquées dans le cadre de la gestion d'environnements ambiants. La sous-section suivante dresse un état des différents travaux sur le déploiement automatique d'applications dans les systèmes distribués.

2.2 Déploiement automatique d'applications dans les systèmes distribués

La notion de déploiement automatique d'applications remonte aux années 90. Les logiciels n'étant alors plus des programmes autonomes fournis par un seul et unique distributeur, mais des ensembles de composants interagissant entre eux, la principale préoccupation de l'époque était donc de savoir comment installer automatiquement des ensembles de composants logiciels répartis sur une machine ou sur un réseau. Carzaniga et Al. [Carzaniga 98] regroupent dans le terme de déploiement toutes les activités

relatives à la mise en production, l'installation, l'activation, la désactivation, la mise à jour et la suppression de composants. Ces composants possédant des dépendances ainsi que des caractéristiques propres ne peuvent pas être déployés dans n'importe quelle ordre ni sur n'importe quelle machine du réseau. Cela peut devenir un vrai casse-tête pour les administrateurs systèmes [Sapuntzakis 03]. C'est pour cela que des outils de résolution des dépendances ont été tout d'abord développés, puis par la suite des outils de configuration automatique de machines.

Papadopoulos et al. [Papadopoulos 03] proposent Rocks, un outil de gestion et de configuration de clusters de machines homogènes à grand échelle. Rocks permet d'installer sur une machine physique un système d'exploitation complet de type GNU/Linux qui fournit les outils de base pour la supervision du nœud. Le processus étant automatisé, il peut être exécuté de manière à configurer un grand nombre de machines. Des mécanismes permettent de produire facilement des configurations customisées pour un sous-ensemble de nœuds.

Kanies et al. [Kanies 06] proposent Puppet, un outil de gestion des configurations de machines permettant de s'abstraire du système d'exploitation sous-jacent. Ceci est permis grâce à la mise à disposition de l'utilisateur d'une API cachant les détails spécifiques aux systèmes d'exploitation. C'est un outil d'automatisation qui permet de définir des fichiers de configuration de manière déclarative et d'exécuter ceux-ci afin de configurer un ensemble de machines.

De nombreux autres travaux basés sur les mêmes principes de création d'applications par composition de composants logiciels réutilisables existent [Schantz 02]. Ils sont la base des travaux actuels du déploiement d'applications dans différents domaines. Les sous-sections suivantes dresseront un état de l'art du déploiement d'applications dans les domaines des services (Sous-Section 2.2.1), dans celui des grilles de calculs (Sous-Section 2.2.2) puis du Cloud Computing (Sous-Section 2.2.3).

2.2.1 Déploiement de services

L'ingénierie logicielle orientée services (SOSE [8]) est un paradigme de programmation qui s'est développé en parallèle de l'ingénierie logicielle orientée composants (CBSE [9]) et avec laquelle elle partage beaucoup de points communs [Breivold 07] comme le besoin de composer les briques de base de chacune des technologies (composants et services). Néanmoins, une fonction fondamentale des architectures orientées services (SOA [10]) est la séparation entre la logique métier des services et la plateforme sur laquelle ces services sont implémentés et déployés. La plupart des travaux dans le domaine des SOA se focalisent donc sur la manière de concevoir, d'implémenter, de composer et de placer ces services [Curbera 05]. Néanmoins, certains chercheurs ont compris l'intérêt de raisonner sur la couche de déploiement de ces services. Nous nous focalisons dans cette sous-section sur ces travaux de déploiement qui, dans un premier temps, se basent sur les concepts et les technologies des approches basées sur les composants et qui

8. Service Oriented SOftware Engineering
9. Component-Based Software Engineering
10. Service Oriented Architecture

s'émancipèrent par la suite de cette technologie afin de proposer des mécanismes de déploiement propres aux SOA.

Donsez [Donsez 07] présentent un moyen de déployer des composants logiciels implémentant des points de contrôle UPnP. Dans ce travail, l'auteur propose trois types de composants appelés « controlets », « bridglets » et « canevas ». Un « controlet » est un composant de contrôle d'un équipement ou d'un service. Celui-ci s'abonne aux variables d'état du composant et est notifié automatiquement lors d'un changement. Il met à disposition de l'utilisateur une interface permettant de superviser l'état de l'équipement ou du service. Un « bridglet » est une passerelle réalisant des conversions d'opération et de notification entre les différents équipements. Enfin, le « canevas » utilise les informations attachées aux « controlets », « bridgelets » mais aussi aux équipements eux-mêmes (les équipements UPnP embarquent leur propre description en XML) afin de réaliser des opérations de courtage, de composition et de déploiement. L'opération de courtage permet de sélectionner le composant le plus adéquat pour gérer un nouvel équipement. L'opération de composition permet de définir des composants correspondant à un ensemble de sous-composants. Enfin, l'opération de déploiement permet de déployer les composants lorsqu'un nouvel équipement est découvert, ou bien lorsque l'utilisateur a besoin de cet équipement. L'auteur met en avant la modulabilité ainsi que la composabilité de son architecture. Modulaire car un composant peut prendre en charge un équipement entier, un sous-équipement, un service d'un équipement ou d'un sous-équipement. Composable car un composant peut-être composé d'autres sous-composants traitant d'un équipement ou d'un service. Le prototype compatible OGSi [11] [Marples 01] réalisé est implémenté en Java et la phase de déploiement consiste simplement au lancement des composants sur une machine [Carzaniga 98].

Un autre travail de Donsez et Al. proposent un démonstrateur d'une plateforme orientée services pour le déploiement d'applications iTV [Donsez 10]. Les télévisions interactives permettent aux développeurs de proposer des applications pouvant s'exécuter sur des intergiciels spécifiques à chaque marque de télévision. Les auteurs proposent ici une plateforme OSGi pour le déploiement à grande échelle et l'exécution automatique d'intergiciels et de composants applicatifs iTV respectant les standards du domaine. Les applications sont modularisées et décrites sous la forme d'un graphe de services. Ces services peuvent être ajoutés ou retirés dynamiquement à l'application pendant son exécution. Peu de détails sont malheureusement donnés. Néanmoins, il est intéressant de noter que les applications sont décrites à l'aide de graphes représentant les services et leurs interactions. Afin de déployer l'application, un moteur de raisonnement doit donc parcourir le graphe de l'application et trouver dans l'infrastructure les services qui respectent les contraintes décrites.

Les travaux d'Edwards et Al. [Edwards 04] s'inscrivent clairement dans la lignée des approches orientées composants. Néanmoins, les auteurs proposent ici un moyen de déployer des services publisher/subscriber (publication/souscription) au sein d'intergiciels orientés composants de type CORBA [Vinoski 97] à l'aide d'une approche dirigée par les modèles (Model Driven Approach). Le CCM [12] spécifie les mécanismes d'implémentation, de composition et de déploiement des composants. Néanmoins, le

11. Open Service Gateway initiaitve
12. CORBA Component Model

défi relevé par les auteurs a été de configurer et de déployer une fédération de services de type publisher/subscriber alors qu'aucun mécanisme ni aucune politique de déploiement n'existait. Avec l'augmentation du nombre de nœuds et d'abonnés, ce type de service peut devenir un véritable goulot d'étranglement. C'est pourquoi il est important de pouvoir lier les différents nœuds en une fédération de services de type publisher/subscriber. Dans ce type d'architecture, les nœuds sont assignés à des hôtes particuliers et les événements reçus par un nœud sont automatiquement propagés aux autres nœuds de la fédération. Dans ces travaux, la question du déploiement est donc de savoir comment déployer les différents services publisher/subscriber afin d'obtenir une fédération efficace ne surchargeant pas le réseau de messages. La propagation des événements est prise en charge par un composant CORBA appelé passerelle. Pour déployer automatiquement les services et les passerelles sur le réseau, une approche MDM [13] est adoptée. Les auteurs utilisent le métamodèle EQUAL [14] [Deng 04] pour modéliser les différents paramètres et contraintes afin de configurer et de déployer des fédérations de services publisher/subscriber à trois niveaux de granularité : (1) le niveau canal de communication ; (2) le niveau passerelle et ; (3) le niveau événement. Ces modèles sont utilisés pour déployer les services et les composants CORBA. Un des avantages de cette approche est l'utilisation d'un métamodèle permettant la validation automatique des modèles générés pour la configuration et le déploiement.

Alors que beaucoup de travaux dans le domaine du déploiement de services visent à sélectionner les services et à les composer pour satisfaire les besoins d'une application [Rao 04, Kichkaylo 04], les travaux de Brossardt et Al. [Bossardt 03] visent à répondre au défi de la configuration de l'infrastructure pour déployer des services, et en particulier la configuration de la couche réseau. Alors que dans la plupart des travaux, ces besoins sont décrits de manière informelle dans des documents, le travail de Brossardt et Al. introduit une nouvelle approche à base de patron de modèles (model-based pattern) afin d'utiliser des mécanismes formels pour trouver la bonne configuration de l'environnement réseau qui supportera l'exécution des applications. Les auteurs identifient trois types de besoins qui doivent être pris en considération :

— Les besoins de la topologie réseau définissent la place d'un composant en fonction des autres.
— Les besoins en terme de ressources spécifient la quantité de bande passante, de mémoire et de puissance processeur des nœuds sur lesquels les composants seront déployés.
— Les besoins fonctionnels déterminent les fonctionnalités de chaque nœud.

La solution proposée a été décomposée en plusieurs étapes :

— L'étape de sollicitation permet d'explorer le réseau à la recherche des différents nœuds sur lesquels les composants pourront potentiellement être déployés.
— L'étape de configuration rassemble les informations récoltées lors de la première phase et choisit les nœuds sur lesquels seront effectivement déployés les composants.
— Enfin, l'étape de déploiement regroupe une phase de requête des nœuds concernés, d'installation des composants et de notification de l'état du déploiement.

La première étape est guidée par des patrons de navigation qui indiquent comment parcourir le réseau à la recherche de nœuds et quels sont les informations à récupérer.

13. Model-Driven Middleware
14. Event QoS Aspect Language

La seconde étape utilise le patron de déploiement spécifique à l'application générée lors de la première étape et essaye de générer les spécifications pour le déploiement d'une instance de l'application en fonction d'un certain nombre de paramètres pré-définis. La nouveauté de ce travail est de fournir une description précise, sous la forme de patrons, des manières de sélectionner les nœuds sur le réseau afin de déployer les composants d'un service.

Les travaux d'Arnold et Al. [Arnold 07] reprennent la même vision que ceux de Bossardt et Al. en utilisant une approche à base de patrons de modèle pour configurer d'infrastructure sur laquelle seront déployés les services. Les besoins des services imposent des contraintes sur la configuration des machines qui les hébergeront. Les patrons de déploiement capturent la structure des solutions en mettant en avant certaines propriétés désirées comme la sécurité, la haute disponibilité ou encore le passage à l'échelle [Redlin 06] afin de supporter des classes d'applications avec différentes caractéristiques. Ces patrons correspondent donc à une topologie d'application. La Figure

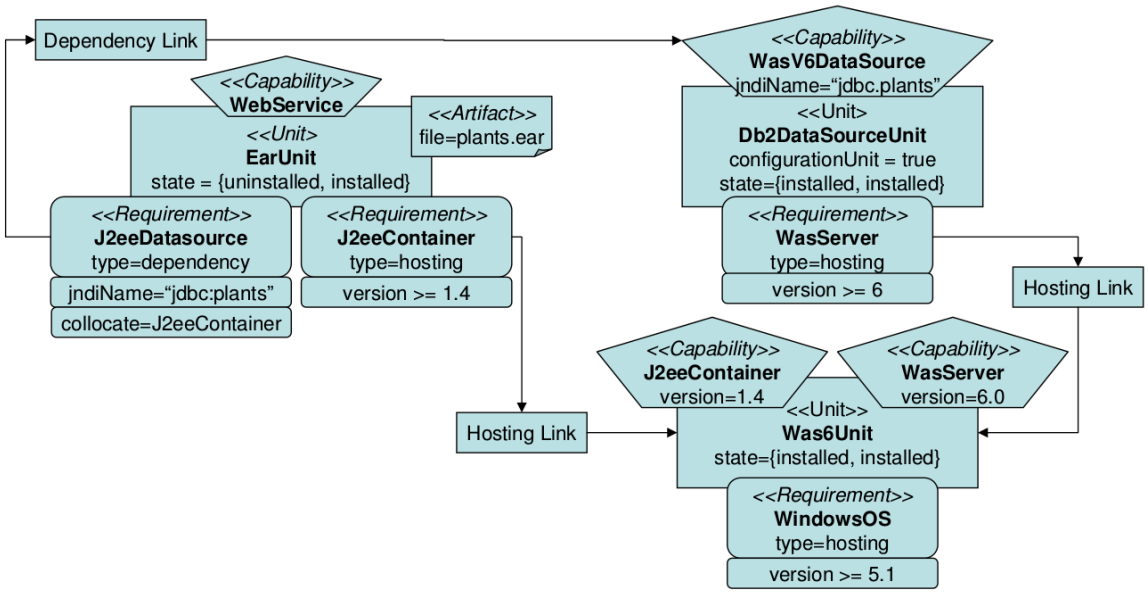


FIGURE 2.4 – Exemple de topologie de déploiement pour une application de type J2EE

2.4 représente un exemple de topologie pour une application de type J2EE. Afin de déployer une application correspondant à un patron particulier, une topologie valide doit être trouvée entre la topologie de l'application à déployer et la topologie cible, représentant l'environnement disponible. Les travaux présentés dans [Arnold 08] détaillent l'algorithme permettant de réifier les patrons de déploiement des applications sur l'infrastructure disponible (Pattern Realization). Les auteurs utilisent un algorithme de graph-matching recherchant un isomorphisme de graphes afin de faire correspondre à chaque élément d'une topologie d'application, une entité dans la topologie de l'infrastructure qui réponde aux besoins. La Figure 2.5 est un exemple du processus de reconfiguration de l'infrastructure pour une application qui installe une base de données sur un système d'exploitation Windows. L'algorithme de graph-matching associe pour chaque unité de la topologie de l'application (DB2System et DB2Instance) une entité de la topologie de l'infrastructure qui répond aux contraintes.

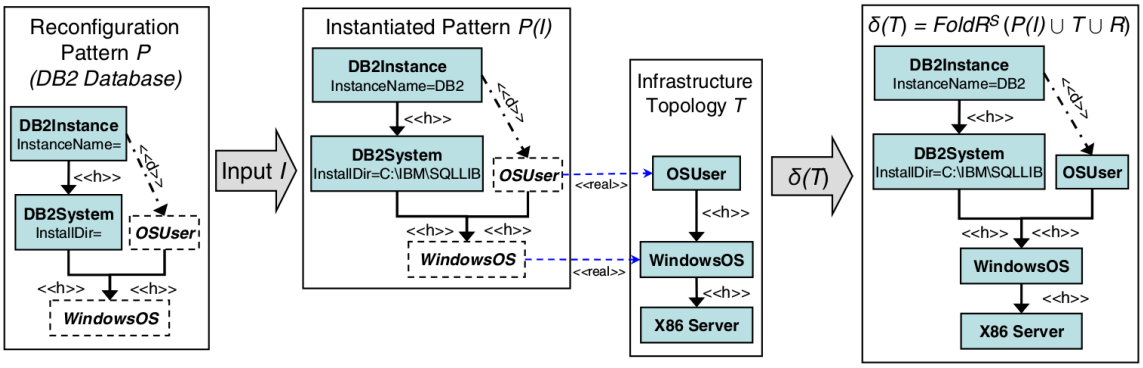


FIGURE 2.5 – Exemple du processus reconfiguration de l'infrastructure pour une application

Basés sur les mêmes idées d'un modèle de description des besoins des applications, les travaux de Kannan et Al. [Kannan 09] présentent CM (Configuration Map), un ensemble de paramètres de configuration pour des intergiciels de déploiement de composants, ainsi que leurs liens. Un paramètre de configuration possède une valeur qui dépend du besoin du composant à déployer, de son environnement et de ses dépendances. Chaque composant du système se voit donc décrit par un ensemble de paramètres de configuration regroupés et organisés au sein d'une CM. Cette CM permet de fournir des mécanismes formels pour configurer les composants à déployer et associer un ensemble d'exceptions qui permettent de déterminer les erreurs de configuration potentielles. Les paramètres de configuration peuvent être obtenus à l'aide d'outils de découverte comme TADDM [15] [Rich 07] ou CCMDB [16] [Change 12]. La Figure 2.6 illustre l'architecture proposée par les auteurs.

Afin de déployer des applications sur une architecture orientée services, les travaux les plus récents proposent des descriptions formelles, souvent à base de patrons avec des architectures orientées modèles, des besoins des applications pour que celles-ci puissent s'exécuter sur une infrastructure avec une certaine qualité de service. Ces descriptions formelles permettent de trouver des mécanismes pour automatiser la configuration de l'infrastructure et le déploiement des applications respectant les contraintes et besoins de celles-ci.

2.2.2 Déploiement automatique sur grilles de calcul

Le déploiement d'applications distribuées sur une grille de calcul est un domaine de recherche majeur du milieu des années 1990 à la fin des années 2000. Un des objectifs à long terme est de fournir une vaste infrastructure de calcul qui soit totalement transparente pour l'utilisateur, à l'image du réseau de distribution électrique [Foster 03]. Originellement effectué à la main par des administrateurs systèmes, le déploiement sur une infrastructure peut être un vrai cauchemar. On peut le définir comme un ensemble de taches d'orchestration : connexion à des nœuds distants, installation et désinstallation de logiciels, configuration des nœuds et des logiciels, lancement ou arrêt de services

15. Tivoli Application Dependency Discovery Manager
16. Tivoli Change and Configuration Management Database

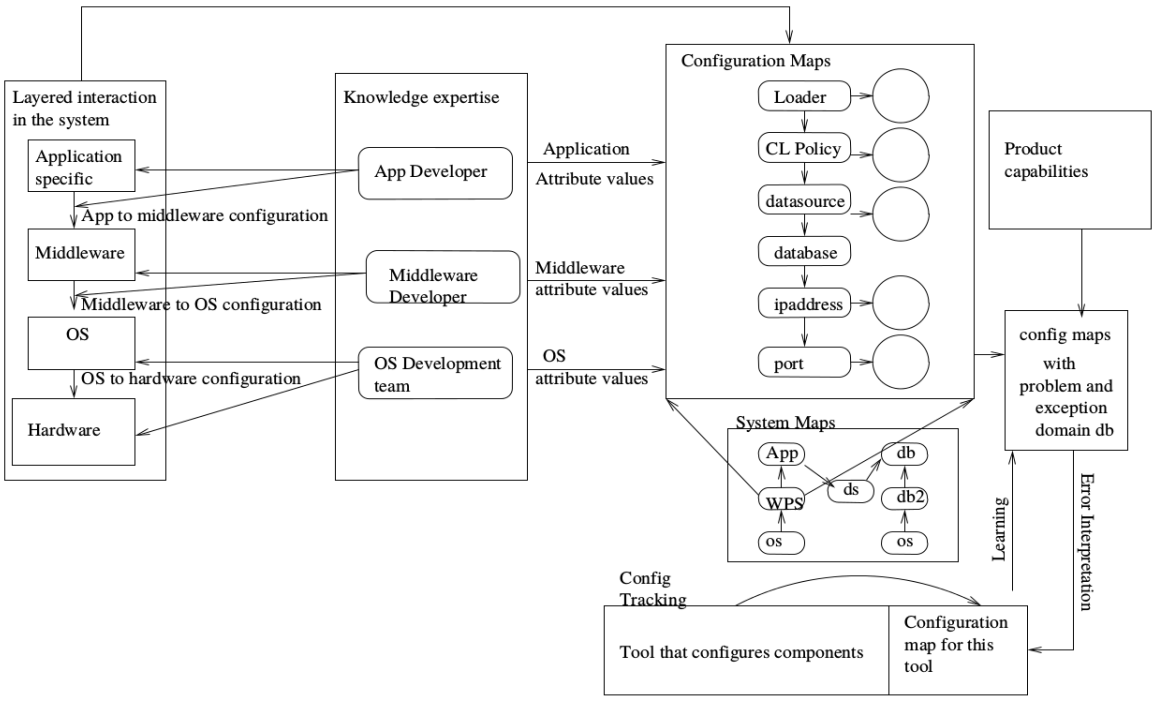


FIGURE 2.6 – Architecture opérationnelle de CM

ou de processus, transfert de données, etc. Le déploiement sur grilles de calcul introduit plusieurs challenges à relever :

— Le premier challenge est celui de la **complexité** des systèmes. Il est courant d'avoir des grilles de plusieurs milliers de nœuds. Dans ces cas, il est impensable d'effectuer un déploiement d'applications à la main. Les tâches de déploiement doivent être automatiquement orchestrées, les dépendances des programmes automatiquement gérées, et la grille doit pouvoir être supervisée et contrôlée.

— Le second challenge est celui de la prise en compte de l'**hétérogénéité** relative des grilles de calcul. L'infrastructure peut, en effet, être composée de nœuds et de ressources possédant des caractéristiques et des puissances différentes. Ces ressources sont certes toutes des entités matérielles de calcul, mais les propriétés de ces entités tels que le système d'exploitation, l'architecture ou les accointances sur le réseau peuvent varier. Les mécanismes de déploiement (SSH, puppet, Telnet, etc.) ainsi que les protocoles de transfert (HTTP, FTP, etc.) peuvent aussi être multiples. Les applications distribuées à déployer peuvent aussi être hétérogènes en terme de technologies utilisées (programmation orientée objets, architecture orientée services, programmation parallèle, etc.), ainsi que de types (serveur d'applications, intergiciel, composant, application pour un intergiciel particulier, exécutable, bibliothèque, machine virtuelle, etc.).

— Le troisième challenge auquel doit faire face le déploiement automatique sur grilles est celui de la **validation** d'un déploiement. Avant de déployer effectivement une application, il faut garantir que le plan de déploiement est cohérent et fiable afin d'éviter les inconsistances. Les conflits de ressources, la validation de dépendances, l'accessibilité des données/ressources font partie de ce problème de validation.

— Enfin, le dernier challenge est celui du **passage à l'échelle**. Une grille de calcul

pouvant comporter plusieurs milliers de nœuds ressources interconnectés, il est important de prendre en compte les limites de ces ressources (nombre de ports disponibles par nœud, bande passante entre deux nœuds, etc.) afin de déployer des millions de nœuds applicatifs travaillant en parallèle.

Afin d'adresser ces challenges, beaucoup de travaux visent à proposer des intergiciels pour le déploiement automatique d'applications distribuées et hétérogènes sur des grilles de calcul. La plupart de ces intergiciels permettent de faciliter le développement, la composition et/ou le déploiement d'applications sur une grille. Gannon et al. [Gannon 02] proposent de catégoriser les utilisateurs de grilles de calcul en trois groupes :

— Le premier groupe correspond aux utilisateurs finaux qui décident de déployer des applications complètes sur une grille à l'aide d'une interface graphique ;
— Le second groupe correspond aux développeurs d'applications qui savent comment les programmer en composant les différents « composants logiciels » ;
— Enfin, le dernier groupe d'utilisateurs sont ceux qui développent ces briques logicielles unitaires qui pourront être ensuite composées par le second groupe pour proposer des applications déployables par le premier.

La plupart des travaux sur les grilles de calcul utilisent des méthodologies orientées composants. C'est le cas de CAA [Gannon 02] avec sa boite à outils CCAT/X-CAT [Bramley 00], CORBA [17] [Vinoski 97, Denis 03a] ou encore Pacido et GridCCM [18] [Pérez 03, Denis 03b].

Baduel et al. [Baduel 06] introduisent une plateforme nommée ProActive pour le développement et le déploiement de composants et d'applications distribuées et parallélisables, ainsi que des méthodologies pour programmer de telles applications. L'approche est ici appliquée aux grilles de calcul, mais les auteurs précisent qu'elle peut être reprise pour le déploiement d'applications dans des domaines variés tel que le calcul distribué sur l'Internet. La plateforme est développée en Java afin de cacher l'hétérogénéité du support d'exécution, contrairement à des langages natifs dont les programmes doivent être compilés en fonction de l'infrastructure sur laquelle ils seront exécutés. Les composants ProActive développés en Java sont décrits en XML à l'aide d'un métamodèle prenant en compte le type de composant, l'ensemble des ports utilisés, la faculté de les composer ou de les paralléliser ainsi que le nombre d'interactions qu'ils peuvent avoir avec d'autres composants. Les applications sont elles-aussi décrites à l'aide d'un langage basé sur XML décrivant les différents nœuds et leur déploiement est effectué en faisant correspondre chaque nœud applicatif à un nœud de la grille de calcul en garantissant que la communication entre les nœuds applicatifs peut être établie et en respectant la politique de confidentialité définie dans la description de l'application.

Le travail de Caron et al. [Caron 06] introduit GoDIET, un outil pour la configuration, le déploiement et le management de processus pour DIET (Distributed Interactive Engineering Toolbox). DIET est un fournisseur de service applicatif (ASP [19]) permettant de faciliter l'accès à des ressources calculatoires sur une infrastructure distribuée. Ce travail s'intéresse aux approches automatiques pour configurer et déployer une hiérarchie d'agents DIET (Master Agents et Local Agents) au travers d'une grille. GoDIET

17. Common Object Request Broker Architecture
18. Grid Corba Component Model
19. Application Service Provider

est lui aussi écrit en Java et les propriétés des ressources de calcul et de stockage sont décrites en XML. À partir de ces descriptions, les fichiers de configuration nécessaires au lancement d'agents DIET sont automatiquement générés en fonction des ressources, une planification du déploiement est dressée à l'aide d'un algorithme de planification de type *steady-state* présenté dans [Caron 04] puis ces plans de déploiement sont exécutés afin de déployer les agents correspondants.

Danelutto et Al. [Danelutto 06] présentent GEA [20], un outil de déploiement automatique d'applications sur différentes structures de grilles conçues pour des applications distribuées consommant de grandes quantités de données. Les applications prises en charge par GEA sont développées à l'aide de l'outil de programmation haut niveau ASSIST [Vanneschi 02]. Les besoins des applications sont décrites à l'aide du format de description ALDL [21]. GEA raisonne ensuite sur ces descriptions afin de planifier le déploiement des applications sur une grille en découvrant les ressources, en sélectionnant les différents programmes à exécuter et en les installant.

Les trois travaux précédents décrivent des intergiciels proposant des méthodologies et des outils pour développer et déployer des applications distribuées sur une grille de calcul. Ainsi, le développeur choisit la plateforme de développement et s'il respecte la méthodologie et utilise les outils et les technologies associés, il pourra déployer automatiquement son application. De nombreux autres travaux (Jade [Bouchenak 06]) sont basés sur les mêmes principes. Malheureusement, tous imposent au développeur leurs outils et leurs technologies.

Néanmoins, certains autres travaux proposent des plateformes plus génériques laissant aux développeurs des composants la liberté d'utiliser les technologies qu'ils souhaitent. C'est le cas de Flissi et al. [Flissi 06, Flissi 08] qui proposent DeployWare, un intergiciel pour déployer des applications sur une grille de calcul. Cet intergiciel est composé d'un meta modèle de type DSML (Domain Specific Modeling Language) pour abstraire le concept de déploiement ; d'une machine virtuelle distribuée s'occupant de raisonner, de valider et de déployer l'application ; ainsi que d'une interface graphique pour superviser et contrôler l'intergiciel.

Lacour et al. proposent ADAGE [22] [Lacour 04b, Lacour 04a] et GADe [23] [Lacour 05]. Ces travaux définissent un métamodèle pour décrire de manière générique les applications afin de les déployer de manière automatique sur une grille de calcul. Le but de ce travail est de pouvoir déployer plusieurs programmes interagissant ensemble sur plusieurs ressources de la grille. L'application est décrite au format XML et contient plusieurs processus. Pour chacun d'entre eux est spécifié le système d'exploitation requis, l'architecture du processeur désiré, un lien vers l'exécutable ainsi que l'ensemble des dépendances. Les interactions entre les processus sont décrites en spécifiant simplement quel processus doit communiquer avec quel autre. Plutôt que d'être spécifique à une plateforme, la description de l'application est ici générique, ce qui permet d'utiliser un moteur de planification lui-aussi générique. Néanmoins, ADAGE ne permet que le déploiement statique des applications.

20. Grid Execution Agent
21. Application Level Description Language
22. Automatic Deployment of Applications in a Grid Environment
23. Generic Application Description model

Ces deux derniers travaux ont pour avantage de proposer un métamodèle permettant de décrire de manière générique les applications à déployer [Coppola 07], alors que les travaux précédents proposent des modèles de description pour des plateformes et intergiciels spécifiques.

Tous les travaux cités dans cette Sous-section reposent en grande partie sur les mêmes principes :
— Une description des applications et des ressources plus ou moins génériques
— Un moteur de planification, potentiellement configurable
— Un moteur d'exécution du plan de déploiement ainsi généré

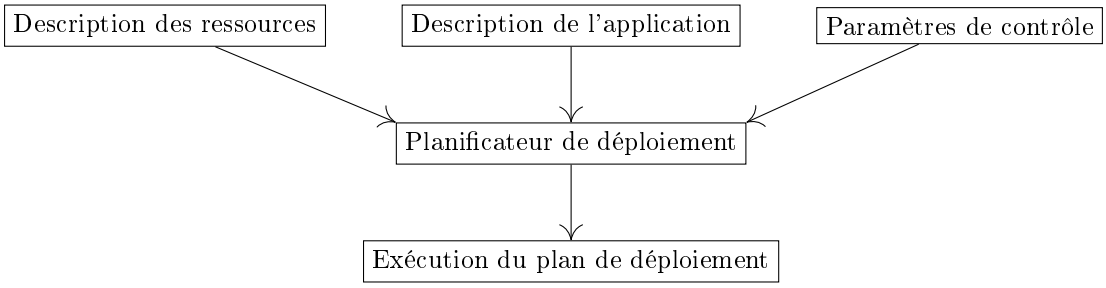


FIGURE 2.7 – Vue générale du processus de déploiement automatique d'applications sur une grille de calcul

La Figure 2.7 illustre le processus général de déploiement automatique d'applications sur une grille de calcul. Néanmoins, ces grilles de calcul contiennent des ressources relativement homogènes ; ce sont toujours des machines de type ordinateur et le déploiement d'applications se résume souvent à télécharger sur ces machines un exécutable et le lancer. Même si le processus général est abstrait et peut être applicable pour les systèmes ambiants, ce n'est pas le cas des différentes méthodes exposées ci-dessus. En effet, dans un système ambiant, il faut prendre en compte l'hétérogénéité du matériel. Les ressources ne sont plus uniquement des ordinateurs, mais aussi des capteurs ou des actionneurs qui peuvent avoir des contraintes très fortes en termes de consommation énergétique ou de bande passante disponible sur les réseaux. Le modèle de description ainsi que les mécanismes de planification et de déploiement sont donc beaucoup plus complexes.

2.2.3 Déploiement dans les nuages

Le Cloud computing (calcul dans les nuages) est devenu un nouveau buzzword depuis l'avènement de l'Internet 2.0 à la fin des années 2000 [Armbrust 09]. Ce nouveau domaine s'inspire clairement du calcul sur grille ainsi que des systèmes distribués en général. La vision en est d'ailleurs la même : les calculs ne s'effectuent plus localement, mais sur des infrastructures externes offrant des services et des outils de calcul et de stockage. Néanmoins, le Cloud computing diffère du calcul sur grille puisque la quantité de données à analyser a augmenté de plusieurs ordres de grandeur. Ainsi, les techniques de traitement massive de données (Big Data), l'évolution des technologies ainsi que l'engouement des industriels pour ces concepts (Amazon, Google, Microsoft, etc.)

font du Cloud computing un nouveau domaine de recherche très prolifique (conférer Chapitre 1.2.2.3).

Il existe beaucoup de manières de définir le Cloud computing [Inc 08]. La définition que nous avons retenue est celle de Foster et al. [Foster 08].

> *A large-scale distributed computing paradigm that is driven by economies of scale, in which a pool of abstracted, virtualized, dynamically-scalable, managed computing power, storage, platforms, and services are delivered on demand to external customers over the Internet.*

Cette définition met en avant le fait que le Cloud computing est intimement lié au paradigme du calcul distribué avec pour particularités (1) de pouvoir être mis à l'échelle sur plusieurs ordres de grandeur, (2) de configurer dynamiquement des services afin de les délivrer à la demande et (3) d'être considéré comme une seule et unique entité abstraite fournissant aux utilisateurs un ensemble de services. La Figure 2.8 illustre ce dernier point. Le client interagit avec une entité abstraite (Cloud computing) afin d'obtenir des services mettant en œuvre un certain nombre de données. Ces dernières sont stockées, produites, consommées et accessibles sur l'Internet au travers d'un système de fichiers distribué [Juve 10].

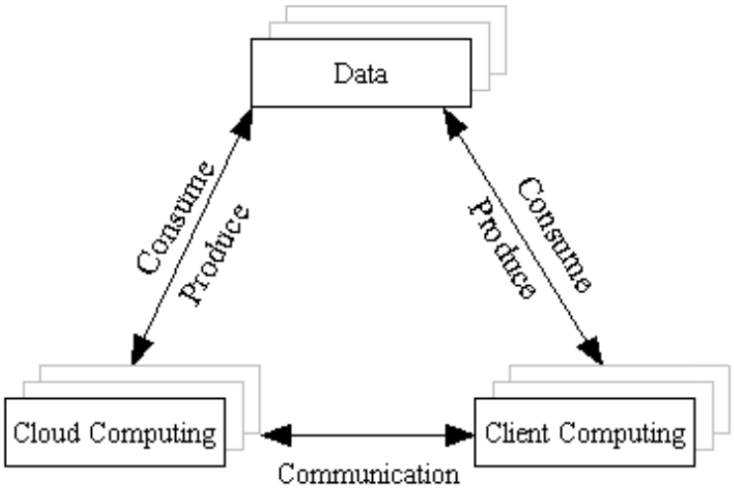


FIGURE 2.8 – Abstraction de l'infrastructure de calcul avec le Cloud (extrait de [Foster 08])

Contrairement aux grilles de calcul pour lesquelles le déploiement d'applications se fait en fonction de la configuration des machines et est effectué de manière statique, le déploiement d'applications pour le cloud computing passe par l'installation et la configuration dynamique de machines virtuelles qui sont mises à disposition de l'utilisateur, lequel a donc un contrôle total de l'environnement logiciel dans lequel ses applications fonctionnent. Un tel système de déploiement doit donc prendre en compte un certain nombre de critères. Juve et al. [Juve 11a] isolent cinq de ces critères :

— Le déploiement automatique d'applications distribuées. Ces applications demandent bien souvent des environnements complexes à mettre en place afin

d'être correctement exécutées. Un service de déploiement doit permettre à un utilisateur de décrire les nœuds et les services requis. À partir de cette description, le système doit pouvoir mettre à disposition l'infrastructure et configurer automatiquement les applications à la demande.
— La prise en compte de dépendances complexes. Afin de déployer ces applications distribuées et de mettre à disposition les nœuds et les services, le système doit être configuré dans le bon ordre en fonction des dépendances de l'application qui peuvent souvent être exprimées à l'aide d'un graphe sans cycle.
— La mise à disposition dynamique (provisionning) des ressources [Jennings 15]. Les besoins de ces ressources pouvant changer dans le temps, le système de déploiement doit permettre à l'utilisateur d'ajouter et de retirer des nœuds dynamiquement pendant l'exécution de l'application. Des algorithmes de mise à disposition élastiques (elastic provisionning algorithms) permettent d'adapter facilement le déploiement en fonction des besoins temps réel des applications [Marshall 10, Sharma 11, Shen 11, Islam 12].
— La prise en compte de plusieurs fournisseurs de services dans les nuages. Comme un seul fournisseur de services ne peut pas forcement fournir suffisamment de ressources pour répondre aux besoins d'une application, le système de déploiement doit être capable de déployer cette application sur des infrastructures fournies par plusieurs fournisseurs de services. Cette capacité de déployer des applications dans les nuages, indépendamment des fournisseurs de services utilisés est appelé Sky Computing [Keahey 09, Petcu 10].
— La supervision des applications. Comme le fonctionnement des services et des applications peut rencontrer un certain nombre de problèmes sur le long terme, la supervision de ces services doit pouvoir renseigner l'utilisateur sur le bon fonctionnement de ses applications. Ces interfaces permettent de faciliter les tests du système et d'avertir l'utilisateur en cas d'incohérence.

Le premier point peut-être illustré par les travaux de Papadopoulos et al. et de Kanies et al. [Papadopoulos 03, Kanies 06] présentés au début de ce chapitre et qui peuvent être considérés comme la base des travaux actuels de déploiement d'applications dans le Cloud.

Le déploiement automatique d'applications dans le Cloud reprend les concepts de configuration automatique et de déploiement d'applications sur des machines physiques présentés dans les deux travaux précédemment cités, mais en configurant cette fois-ci des machines virtuelles qui seront le support des applications de l'utilisateur. Des moteurs de planification de services basés sur des SLA[24] ont donc été développés. De nombreux algorithmes et heuristiques de planification ont été proposés par Cao et al. [Cao 09], Lee et Al. [Lee 10], Garg et Al. [Garg 10] ou encore par Pandey et al. [Pandey 10].

L'un de ces algorithmes présenté par Emeakaroha et Al. [Emeakaroha 11] a été intégré à LoM2HiS [Emeakaroha 10, Emeakaroha 13], une plateforme de supervision de ressources pour le Cloud. Cette plateforme intervient sur trois niveaux différents d'allocation de ressources : (1) le niveau infrastructure (IaaS[25]), (2) le niveau plateforme

24. Service Level Agreements
25. Infrastructure as a Service

(PaaS [26]) et (3) le niveau applicatif (SaaS [27]). La Figure 2.9 représente le modèle d'allocation et de déploiement de ressources. Un utilisateur émet une requête de déploiement

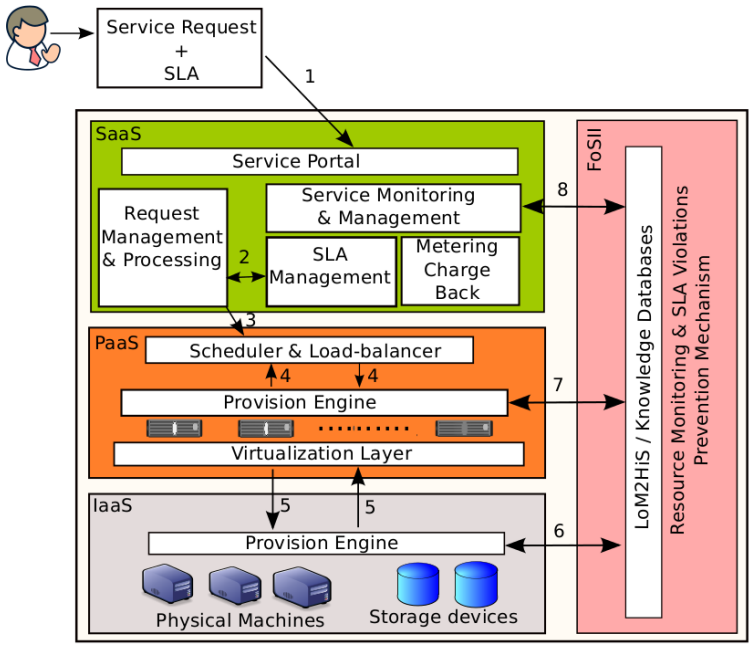


FIGURE 2.9 – Architecture de la plateforme LoM2HiS (extrait de [Emeakaroha 11])

de services au composant « Service Portal » de la couche SaaS (1). Ce dernier va valider la requête (2) avant de la passer à la couche PaaS pour planifier un déploiement (3). Ce planificateur va mettre à disposition des machines virtuelles grâce au composant « Provision Engine » (4). Ce dernier interagit avec la couche IaaS afin de déployer effectivement les machines virtuelles sur les ressources physiques. Cet architecture permet d'abstraire les couches infrastructure et plateforme à l'utilisateur tout en respectant les cinq critères présentés ci-dessus.

Brandtzaeg et Al. présentent CloudML [Brandtzæg 12], une plateforme de déploiement d'applications pour le Cloud inspirée des approches par composants. Différents types d'applications déployables sont représentés à l'aide de topologies de déploiement décrites sous la forme de graphes. Chaque nœud requis est décrit (nombre de processeurs, taille de la mémoire, type de système d'exploitation, etc.) et les besoins de l'application sont spécifiés à l'aide de modèles. Ceci permet d'utiliser une approche dirigée par les modèles pour mettre à disposition les ressources et déployer les applications [Ferry 13]. Cette plateforme, toujours en cours de développement, semble rencontrer un vif succès et est utilisée comme base dans différents projets [Bergmayr 15].

Fan et al. présentent une méthode pour le déploiement automatique d'applications à base de topologies [Fan 12a]. Cette approche originale génère automatiquement les topologies logiques (liens de communication au sein d'une application distribuée) et les topologies physiques (liens entre les nœuds de l'infrastructure). Ces topologies serviront au déploiement des applications. Les patrons de communication des applications

26. Platform as a Service
27. Software as a Service

sont découverts automatiquement à l'aide d'algorithmes de clustering multi-échelles [Noack 09]. La topologie logique d'une application est générée à partir de ces patrons de communication. La topologie physique quant à elle est découverte automatiquement grâce à une méthode de type « spectral clustering-based » [Fan 12b]. Une fois les topologies logiques et physiques obtenues, chaque partie de l'application est mappée à un cluster de nœuds de l'infrastrucure avant de pouvoir effectuer le déploiement de l'application.

Juve et al. [Juve 11a, Juve 11b] proposent Wrangler, un système permettant la configuration dynamique du déploiement de services en fonction des besoins des utilisateurs exprimés en XML. La Figure 2.10 représente l'architecture de Wangler. Quatre

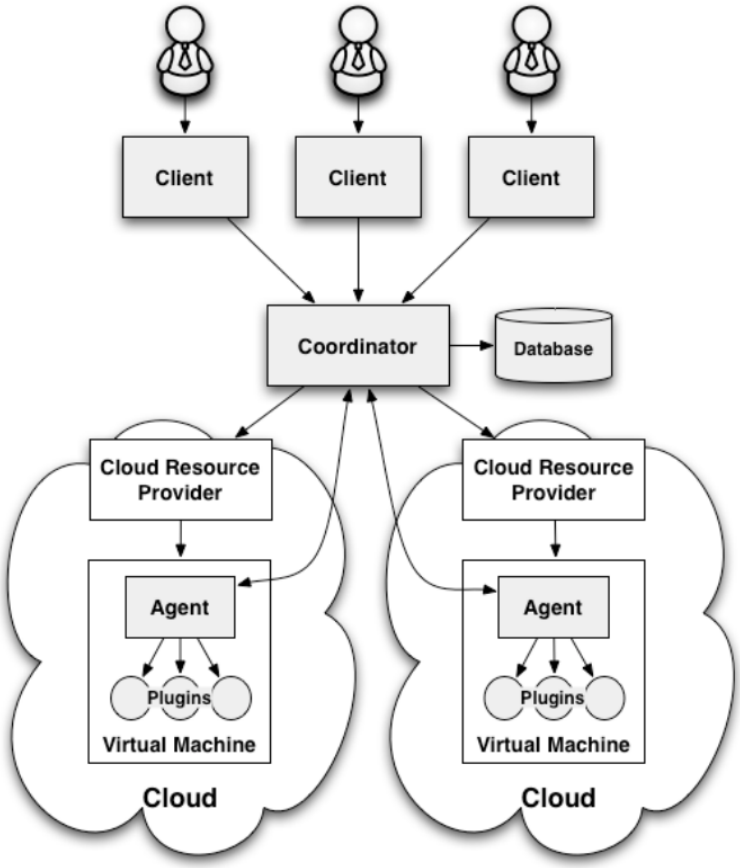


FIGURE 2.10 – Architecture du système de déploiement Wrangler (extrait de [Juve 11a])

composants ont été isolés : les clients, le coordinateur, les agents et les plugins.

— Les clients s'exécutent sur les machines de l'utilisateur et permettent d'envoyer des requêtes au coordinateur, de démarrer, interroger ou annuler un déploiement.
— Le coordinateur est un service web qui supervise le déploiement d'une application. Celui-ci accepte les requêtes du client et met à disposition de celui-ci des nœuds du cloud , collecte les informations quant à la configuration de ces nœuds (c'est à dire installe une machine virtuelle) et au déploiement de l'application et aide à la configuration de cette dernière.
— Les agents sont exécutés sur chacun des nœuds du cloud mis à disposition et permettent de superviser et de configurer la machine virtuelle correspondante.

— Enfin, les plugins sont des scripts développés par l'utilisateur et invoqués par les agents qui définissent le comportement des nœuds.

Le processus de déploiement est ici assez simple. Un client envoie au coordinateur une requête contenant la description en XML de tous les nœuds qui doivent être lancés. Cette description contient le type de fournisseur de service, la référence de l'image de la machine virtuelle à installer, une description des plugins à invoquer et à exécuter ainsi qu'un certain nombre d'options de configuration. Le coordinateur va, à partir de cette description, valider la requête puis contacter les fournisseurs de services afin de mettre à disposition de l'utilisateur les nœuds demandés. Une fois ces nœuds mis à disposition, les machines virtuelles sont installées et les agents sont lancés. Ces agents configurent la machine virtuelle et notifient le coordinateur qui va vérifier la cohérence du nœud. A partir de ce moment, l'agent peut invoquer et exécuter les différents plugins puis superviser régulièrement l'état du nœud.

Le besoin de prendre en compte l'hétérogénéité relative de l'infrastructure d'une grille de calcul a migré vers des problématiques d'allocations dynamiques de machines virtuelles. L'hétérogénéité n'est maintenant plus un souci puisqu'il suffit de configurer la bonne machine virtuelle pour faire fonctionner les applications et les services désirés. On remarque que les travaux dans le cloud computing proposent toujours des métamodèles pour décrire les besoins des applications ou l'infrastructure disponible [Ferry 13]. Par contre, le caractère dynamique du Cloud a permis de développer l'aspect distribué des solutions dans lesquelles des entités indépendantes ont des responsabilités et des rôles unitaires précis. Ces entités coopèrent ensemble, souvent en mode requête/réponse, afin de déployer et de configurer l'environnement sur lequel s'exécuteront les applications de l'utilisateur. De plus, les applications qui étaient souvent conçues à l'aide d'approches orientées composants ont progressivement laissé la place à une approche orientée services [Wei 10].

2.2.4 Synthèse

Nous avons présenté dans cette section différents travaux visant à déployer des applications sur une infrastructure matérielle composée de machines de type ordinateurs. Un certain nombre d'architectures et de technologies ont été développées afin de venir sélectionner dans l'infrastructure les machines sur lesquelles déployer les applications. La quasi totalité des travaux présentés décrivent de manière plus ou moins poussée l'infrastructure ainsi que les besoins des applications afin de pouvoir raisonner et configurer automatiquement le système.

Néanmoins, tous ces travaux ont des lacunes lorsque l'on considère leur utilisation pour le déploiement d'applications ambiantes sur une infrastructure IoT. Les clusters de machines étant souvent composés d'ordinateurs plus ou moins homogènes, la plupart des travaux ne prennent pas en considération l'hétérogénéité matérielle et logicielle, ou du moins, pas de manière suffisamment poussée. D'autres travaux ne s'attaquent pas aux problèmes de la confidentialité. Et enfin, quelques travaux proposent des solutions centralisées qui ne sont pas passables à l'échelle pour des applications réelles d'IAm.

Bien que ces travaux ne soient pas adaptés au déploiement d'applications distri-

buées dans un environnement ambiant, un certain nombre d'étapes du processus de déploiement peuvent être identifiées :

— la modélisation de l'infrastructure disponible
— la modélisation des besoins des applications
— l'utilisation d'algorithmes pour déterminer les plans de déploiement des applications
— l'exécution de ces plans pour déployer effectivement les applications

2.3 Représentation des connaissances

Comme nous venons de le voir dans la section précédente, la modélisation des environnements ambiants est une partie importante à prendre en compte si l'on souhaite pouvoir déployer des applications dans des environnements ambiants tout en considérant l'hétérogénéité des technologies utilisées. Décrire les applications à déployer ainsi que l'infrastructure matérielle disponible est donc indispensable. Dans cette section, nous présentons différents formalismes de représentation des connaissances ainsi que les mécanismes de raisonnement associés.

La représentation des connaissances (KR [28]) est un point crucial du domaine de l'Intelligence Artificielle. Elle a pour but de comprendre la nature de l'intelligence et de la pensée afin de permettre à des ordinateurs de reproduire certains comportements considérés comme humain. Pour cela, les travaux du domaine de la représentation des connaissances visent à proposer des moyens pour encoder symboliquement, ou non, les connaissances humaines ainsi que des mécanismes de raisonnement automatique permettant de générer de nouvelles connaissances de manière informatique. Le formalisme de représentation, le langage d'encodage ainsi que les méthodes de raisonnement sont donc fortement liés. Les systèmes basés sur ces modèles calculatoires sont appelés systèmes à base de connaissances (Knowledge-based systems) [Van Harmelen 08].

Les connaissances modélisables sont très variées. Elles peuvent aller de la modélisation d'une expertise humaine dans un certain domaine, à la représentation du monde tel qu'il est perçu au travers de l'ensemble des connaissances humaines. Il est possible de décrire ces connaissances à l'aide de faits, de règles, de contraintes, d'ontologies... Une ontologie fournit une représentation symbolique, d'objets, de classes, de propriétés et de relations alors que les faits, règles et contraintes représentent des structures calculatoires construites à partir de termes ontologiques [Chein 08]. Certaines méthodes peuvent être utilisées pour modéliser une classe de problèmes particuliers. C'est le cas par exemple du HTML [29] qui est un langage permettant de décrire du contenu d'un document web. D'autres font office de langages de description universelles (GPML [30]) permettant de modéliser n'importe quel type de connaissances, dans la limite d'expressivité de la méthode. C'est le cas par exemple de UML [31] qui est un métamodèle permettant de modéliser les connaissances quelque soit le domaine d'application.

28. Knowledge Representation
29. Hypertext Markup Language
30. General Purpose Modeling Language
31. Unified Modeling Language

Dans cette section, je fais un état de l'art de certains aspects de la représentation des connaissances. Je commence à la Sous-Section 2.3.1 par introduire la représentation des connaissances et définir les termes les plus importants. Dans la Sous-section 2.3.2, je dresse un portrait des différents formalismes et paradigmes permettant de représenter les connaissances ainsi que les méthodes de raisonnement associées. Enfin, à la Sous-section 2.3.3, je présente rapidement ce qu'est le web sémantique et liste les langages de description les plus utilisés.

2.3.1 De la manière d'organiser les connaissances

Avant de parler des différents formalismes pour représenter et raisonner sur les connaissances, ainsi que des langages associés, il est important de comprendre comment il est possible d'organiser un ensemble de connaissances. Cette section a pour but d'introduire brièvement les différentes notions couramment utilisées dans le domaine, telles qu'une méta donnée, une taxonomie, une ontologie ou encore un métamodèle. Ces termes peuvent sembler évidents aux chercheurs du domaine ; il me semble néanmoins important de faire ce rappel afin lever toute ambiguïté possible.

La manière la plus simple d'organiser un ensemble de données est d'attacher à chacune des données un ensemble d'autres données qui permettent de la caractériser. Ces « données à propos des données » sont appelées **méta données**. Par exemple, le titre ou l'interprète d'un morceau de musique sont des méta données du morceau en lui-même – le contenu audio étant la donnée. Les méta données sont la base de la plupart les systèmes de recherche d'informations et permettent d'indexer un ensemble de données selon certains critères prédéfinis. Ces critères prédéfinis correspondent aux types d'informations caractérisant la donnée. On appelle **vocabulaire**, l'ensemble de ces termes explicitement énumérés. Les termes « titre », « compositeur » et « interprète » sont un exemple de vocabulaire des méta données caractérisant un morceau de musique. Ainsi, l'on s'aperçoit qu'il existe déjà plusieurs niveaux d'abstraction : (1) la donnée (le signal audio du morceau de musique) ; (2) la ou les méta données caractérisant la donnée (*The Great Gig in The Sky* des Pink Floyd) ; (3) le vocabulaire possible des méta données caractérisant la donnée (titre et interprète).

Attacher des méta données à des objets est souvent suffisant pour organiser les documents d'une bibliothèque et faciliter les recherches pour localiser un document si un vocabulaire précis est défini. Néanmoins, certaines applications ont besoin de pouvoir organiser de manière hiérarchique les connaissances. C'est le cas d'un blog dans lequel les articles (qui représentent les connaissances à organiser) sont rangés dans différentes catégories et sous-catégories. C'est aussi le cas des morceaux de musique qui peuvent être catégorisés selon leur genre (Jazz, Rock, Metal, etc.). L'organisation hiérarchique des catégories et des sous-catégories entre elles est appelée une **taxonomie**. Une taxonomie est donc à la fois une méthode de classification de concepts décrivant les instances d'un monde particulier et l'ensemble des termes d'un vocabulaire organisé hiérarchiquement. Les relations entre les concepts sont restreintes à la relation d'héritage qui permet une organisation hiérarchique du vocabulaire. Un morceau de musique peut être caractérisé par un genre de musique qui correspond donc à une méta donnée du morceau. Cette méta donnée ne peut prendre qu'une valeur choisie parmi un ensemble

prédéfini qui, s'il est organisé de manière hiérarchique, est alors appelé taxonomie. Le genre Jazz regroupe le jazz New Orleans, le Swing ou encore la Free jazz alors que le Métal regroupe différents sous-genres tels que le Heavy metal, le Speed metal ou le Folk metal lui-même divisé en Métal celtique, Pirate metal ou encore Métal médiéval. On peut donc remarquer différents niveaux de représentation des données : (1) Une donnée peut-être caractérisée par des méta-données dont les types sont choisis parmi un vocabulaire spécifique (titre, interprète, genre) ; (2) les valeurs possibles d'une méta donnée peuvent à leur tour être choisies à partir d'un vocabulaire organisé sous la forme d'une taxonomie. La Figure 2.11 représente une taxonomie possible des genres musicaux caractérisant un morceau de musique.

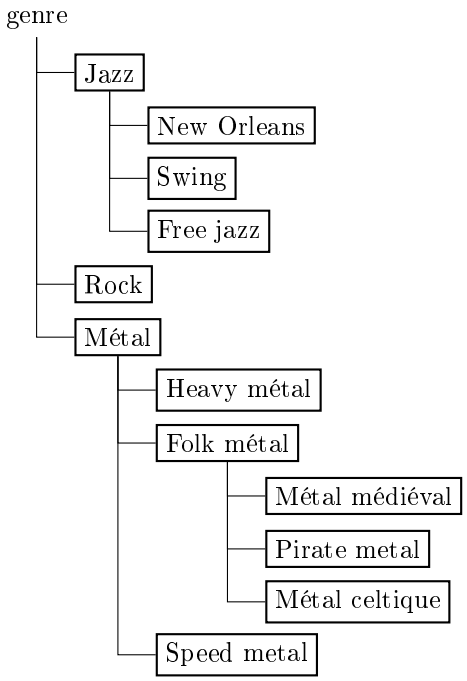


FIGURE 2.11 – Taxonomie (incomplète) des genres musicaux

Le fait de regrouper certaines connaissances au sein de catégories permet de faciliter la proposition de connaissances proches de celle sélectionnée – il est par exemple possible de rechercher tous les morceaux de musique du genre Swing. Afin de permettre plus d'expressivité dans l'organisation hiérarchique de ces connaissances, il est possible d'ajouter différents types de relation entre les termes d'un vocabulaire. Ainsi, une taxonomie qui ne possède qu'une relation implicite d'héritage, se transforme alors en **thésaurus** qui possède plusieurs types de relation explicitement définis. Ces relations sont définies par la norme ISO 25964[32]. On y trouve des relations de type (1) hiérarchique comme BT (Broader term) et NT (Narrower term) permettant de faire référence à des termes respectivement génériques et spécifiques au sein de la hiérarchie, ou encore des relations d'instanciation ou de partitionnement ; (2) associatif comme RT (Related term) qui permet de mettre en relation (composition, causalité, localisation, etc.) deux termes d'une même hiérarchie ; (3) d'appartenance à un groupe de concepts, ce qui permet de définir des micro-thesaurus et de les lier entre eux. D'autres

32. ISO 25964 : The international standard for thesauri and interoperability with other vocabularies

relations annexes sont aussi définies comme SN (Scope note) permettant de décrire de manière informelle un terme du thésaurus. La définition d'un thésaurus plutôt que d'une simple taxonomie permet d'améliorer grandement la robustesse des recherches de données grâce à sa plus grande expressivité pour organiser un ensemble de données.

À la différence d'une taxonomie ou d'un thésaurus, une **ontologie** définit un ensemble de vocabulaire (de manière formelle ou non), non plus pour organiser les connaissances, mais pour les décrire. Une ontologie sert donc à représenter un domaine spécifique et doit pouvoir répondre à la question : « Quel ensemble de choses existe dans un domaine d'application ? », ou de manière plus générale : « Comment peut-on représenter un domaine de connaissances ? ». Une ontologie contient donc un ensemble de concepts permettant de caractériser les connaissances, un ensemble de types de relations sur un ensemble de concepts permettant de définir des relations entre les connaissances, ainsi qu'un ensemble de propriétés caractérisant les instances d'un concept ou d'une relation. La création d'une ontologie passe par un langage de représentation d'ontologies qui permet de définir les ensembles de concepts, de relations et de propriétés de celle-ci.

L'ensemble des connaissances représentant une partie du monde réel est appelé **modèle** du monde. Ainsi, « Un ordinateur composé d'un processeur intel i7 et de 8Go de mémoire vive de type DDR3 » constitue un modèle de l'ordinateur sur lequel ce manuscrit a été rédigé. La manière de construire un modèle (les règles et les contraintes à respecter) constitue un **métamodèle**. La phrase « Le concept d'ordinateur possède un lien de composition avec les concepts de mémoire vive et de processeur. Le concept de mémoire vive possède une propriété "type" pouvant prendre comme valeur "DDR2" ou "DDR3" » définit le métamodèle à partir duquel le modèle précédent a été exprimé. Il est intéressant de noter qu'un métamodèle valide est une ontologie, mais que toutes les ontologies ne sont pas forcement des métamodèles. Il est possible de continuer à s'abstraire en créant des méta-métamodèles permettant de décrire des métamodèles. Ces méta-métamodèles sont en réalité de simples métamodèles. Ainsi, un métamodèle peut être vu de trois manières différentes. Ce peut être :

— un ensemble de règles utilisées pour construire des modèles
— un modèle qui décrit un domaine d'intérêt
— une instance d'un autre modèle

Par exemple, l'OMG[33] a développé le standard MOF[34] qui s'intéresse à la représentation de métamodèles et à leur manipulation. Ainsi, MOF est un méta-métamodèle (ou simplement métamodèle) spécifiant la manière de construire des métamodèles. MOF étant un métamodèle, il s'auto-décrit. Le métamodèle UML est un exemple de métamodèle exprimé à l'aide de MOF et qui permet, entre autre, de modéliser les architectures des applications informatiques orientées objet. Le diagramme de classe d'une application correspond donc au modèle qui décrit l'architecture réelle de l'application. La Figure 2.12 représente un exemple de différents niveaux d'abstraction de la couche MOF à la couche du monde réel. Le nombre de couches est de quatre dans cet exemple, mais il peut-être plus important, MOF permettant de générer récursivement n'importe quel nombre de niveaux de métamodèle.

33. Object Management Group
34. Meta-Object Facility

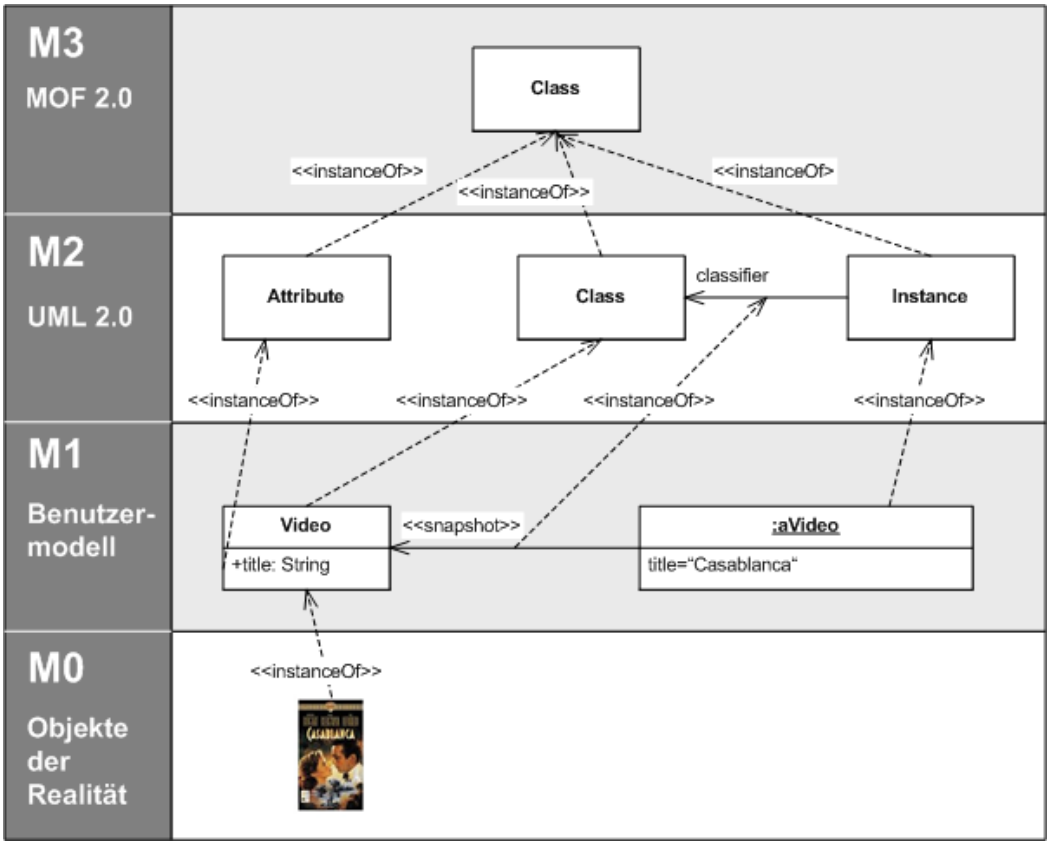


FIGURE 2.12 – Exemple d'une modélisation du monde réel basée sur MOF

2.3.2 Formalismes et paradigmes de représentation des connaissances

Dans cette sous-section, nous introduisons différentes méthodes de représentation des connaissances ainsi que les mécanismes de raisonnement associés. Ce seront les méthodes à base de graphes que nous avons choisi d'utiliser pour les travaux présentés dans la seconde partie de ce manuscrit. La section 2.3.2.1 introduit différentes manières de décrire les connaissances à l'aide de graphes et de raisonner dessus grâce à des méthodes de projection de graphes. La Sous-Section 2.3.2.2 présente certaines méthodes de représentation des connaissances à base de graphes pour lesquelles un langage formelle à base de logique descriptive existe. Nous clôturons cette partie sur les formalismes par la Sous-Section 2.3.2.3 qui présentera succinctement un moyen de décrire les connaissances en spécifiant un ensemble de contraintes. Nous expliquerons dans la synthèse pourquoi cette dernière approche, bien que pertinente pour modéliser notre problème, n'a pas été choisie pour la suite de nos travaux.

2.3.2.1 Premiers pas vers la représentation à base de graphes

En parallèle du développement des formalismes de représentation à base de logique se sont développées dans les années 1960 les représentations sémantiques à base de graphes. La première personne à avoir introduit une notation à base de graphes pour représenter des concepts ainsi que leurs liens est Richens [Richens 56] avec ses ré-

seaux sémantiques. Un réseau sémantique est un graphe labellé et orienté, dans lequel les nœuds correspondent à des concepts et les arcs à des relations liant ces concepts entre eux. La première implémentation d'un réseau sémantique est due à Masterman [Masterman 61] qui l'a utilisée pour extraire la sémantique de messages afin de pouvoir faire de la traduction automatique. Les types de relations possibles sont spécifiés et souvent associés à certains mécanismes. C'est par exemple le cas du lien « is-a » qui permet souvent de lier une instance à une classe et une classe à une super classe, ou la relation « subclass » qui permet de lier hiérarchiquement un ensemble de classes entre elles. Ces relations introduisent un mécanisme d'héritage permettant de transférer certaines autres relations le long de celles-ci. La Figure 2.13 introduit un exemple

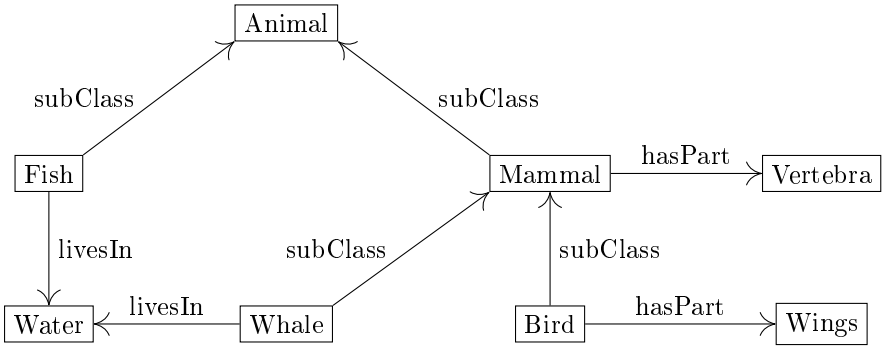


FIGURE 2.13 – Exemple de graphe sémantique

de réseau sémantique. La relation « subClass » est une relation d'héritage qui transmet aux successeurs les autres relations telles que « hasPart » et « livesIn ». Ainsi, un oiseau étant un mammifère, celui-ci possédera une colonne vertébrale.

Les réseaux de corrélation sont des réseaux sémantiques dans lesquels il est possible d'utiliser 56 types de relations introduits dans les travaux de Ceccate [Ceccato 60]. Hays décrit les graphes de dépendances [Hays 64] qui permettent de formaliser la notation introduite par le linguiste Tesnière [Tesnière 59]. Un exemple actuel de réseau sémantique est le web sémantique lui-même [Berners-Lee 01, Hendler 08].

De manière générale, les graphes ont une structure qui permet de proposer des algorithmes de raisonnement, de recherche, d'indexation ou de pattern matching. Ces algorithmes sont basés sur le problème de graph-matching.

Le graph-matching exact fait référence à différentes variations du problème. Le problème de l'homomorphisme de graphe est la plus faible forme de matching ; seuls les liens entre les nœuds doivent être respectés. Les autres problèmes de graph-matching ajoutent des hypothèses au problème général. Le monomorphisme de graphe est un morphisme injectif dans lequel chaque nœud du graphe source est projeté sur un nœud distinct dans le graphe destination. L'épimorphisme de graphe est un morphisme surjectif dans lequel chaque nœud du graphe cible est l'image d'un ou plusieurs nœuds du graphe source. L'isomorphisme de (sous-)graphe est un morphisme bijectif, c'est à dire une relation d'équivalence entre un graphe source et un (sous-)graphe cible. L'automorphisme de graphe est un isomorphisme d'un graphe sur lui-même. Il représente une série de permutations des nœuds du graphe qui respecte les liens d'adjacences entre ceux-ci. Le problème de K-coloration de graphe peut être ramené à un problème

d'homomorphisme du graphe à colorer dans un graphe K-complet. Mis à part l'isomorphisme de graphe (et donc l'automorphisme) pour lequel cela n'a pas encore été prouvé, les autres formes de morphisme sont des problèmes NP-complets [Conte 04], c'est à dire de complexité asymptotiquement exponentielle. Si l'on considère N_H comme le nombre de nœuds d'un graphe destination et N_G, le nombre de nœuds d'un graphe source, il existe potentiellement $N_H^{N_G}$ différentes fonctions à tester. Vérifier si une fonction est un morphisme est de complexité $\Theta(|E_G| + N_G)$.

De nombreux algorithmes et variantes existent pour résoudre ces différents morphismes [McKay 14]. Le plus connu est probablement celui présenté par [Ullmann 76] qui est une variation de l'algorithme de Corneil [Corneil 70] utilisant une technique d'exploration en profondeur et de backtrack. Il s'agit d'un algorithme branch and bound qui explore en profondeur le graphe source et essaye d'associer pour chacun de ses nœuds un nœud dans le graphe destination, tout en suivant les liens entre les nœuds. Un processus de backtrack est utilisé lorsque l'algorithme atteint un point d'inconsistance. Les améliorations de cet algorithme de matching, comme celle proposée par [Cordella 01], consistent en l'ajout d'une heuristique pour l'exploration du graphe. Kim et Al. [Kim 91] ainsi que de Tasi et Al. [Tsai 79] proposent tous deux des heuristiques en fonction de leur domaine d'application. Un autre algorithme a été proposé par [Larrosa 04]. Ce travail exploite le fait qu'un homomorphisme de graphe peut être reformulé comme un problème sous contraintes. Ce problème est ensuite résolu à l'aide d'un moteur de résolution de contraintes classique. [Messmer 00] présente une variante des techniques de matching pour les systèmes experts. Cet algorithme se base sur une décomposition récursive des graphes en sous-graphes. C'est une méthode particulièrement efficace pour correspondre un graphe donné avec une base de graphes pour lesquels la décomposition a pu être calculée en amont. Les travaux de Hoffman [Hoffmann 82] proposent des algorithmes de graph matching exacts qui s'exécutent en un temps polynomial pour les graphes de degré borné. Enfin, [Babai 15] propose un algorithme qui résout le problème d'isomorphisme de graphe en un temps quasi-polynomial.

D'autres algorithmes comme celui de Jong [De Jong 89] sont des algorithmes non exacts qui ne garantissent pas de trouver une solution si elle existe, mais qui ont l'avantage de s'exécuter en un temps polynomial.

Enfin, avec le développement du « big data », des réseaux sociaux et des données massives organisées sous la forme de graphe, des travaux très récents proposent des algorithmes qui résolvent le problème de graph matching de manière distribuée. Les travaux de Ma et Al. [Ma 11, Ma 12] sont les premiers à proposer de tels algorithmes. Leur solution est basée sur le concept de simulation de graphe (graph simulation) [Henzinger 95]. Un graphe de données est distribué sur plusieurs machines. Pour récupérer un sous-graphe de données en fonction d'un patron précis, un « coordinateur » va envoyer ce patron à un ensemble de machines. Ces machines vont rechercher dans leurs sous-graphes de données s'ils peuvent faire correspondre une partie du patron et retourne le résultat au coordinateur. Une fois que ce dernier a obtenu les réponses des machines, il essaye de fusionner les sous graphes afin de trouver une projection du patron sur le graphe de données. Dans ce cas, les données à matcher transitent librement entre les machines afin que le coordinateur possède toutes les informations pour trouver une projection. Cette approche ne permet donc pas d'inclure des mécanismes permettant de limiter la diffusion des informations relatives aux graphes. Un autre

algorithme a été proposé par Fard et Al. [Fard 14] reposant sur une approche centrée sur les nœuds du graphe.

Ces méthodes de représentation des connaissances à base de graphes ont servi à décrire des structures relationnelles du langage naturel et ont été énormément utilisées pour le développement des bases de données ainsi que des langages de requêtes associés. Les représentations présentées dans cette sous section sont des représentations structurées des connaissances mais sans sémantique formelle. La sous-section suivante introduit des formalismes de représentation des connaissances à base de graphes qui sont logiquement fondés.

2.3.2.2 Représentations à base de graphes et logique descriptive

Les réseaux sémantiques permettent de décrire des connaissances sous la forme de graphes et d'utiliser certains algorithmes de graph matching pour raisonner sur ces graphes. Néanmoins, la description de ce type de graphe n'était pas formalisée et des chercheurs comme Woods ont soulevé des questions telles que « Qu'est-ce qu'un lien ? » ou « Qu'est-ce qu'un nœud ? » [Woods 75]. Ce n'est que dans les années 70 que ces graphes se sont vu équipés d'une sémantique formelle à base de logique.

Les premières propositions ont été lancées par Brachman qui présente ses réseaux d'héritage structurés (Structured inheritance networks) ainsi qu'une première implémentation d'un système à base de logique descriptive nommé KL-ONE [Brachman 79] permettant, à partir de connaissances représentées sous la forme de graphes, d'inférer de nouvelles connaissances. Cette logique descriptive est plus explicite que la logique des propositions et peut être traduite à l'aide d'un sous ensemble de la logique des prédicats du premier ordre qui, en général, est décidable. La logique descriptive modélise les concepts (prédicats unitaires), les rôles (prédicats binaires), les individus (constantes) et leurs relations.

Peirce développa dans les années 1980 trois systèmes de notations algébriques, appelés graphes existentiels [Peirce 80, Peirce 85], permettant de modéliser sous forme de graphes des formules de différents types de logique, dont la logique des prédicats du premier ordre. Les graphes conceptuels quant à eux ont été introduits par Sowa [Sowa 76] pour modéliser le langage naturel et faire correspondre des questions en langage naturel et des assertions à des bases de données relationnelles. Ce sont des graphes bipartites composés de concepts et de relations reliés entre eux auxquels l'on ajoute une couche de logique descriptive [Sowa 83]. Les sémantiques formelles des graphes existentiels et conceptuels sont très proches et ont été regroupées au sein du standart ISO Commom Logic [35] basé sur la logique du premier ordre. La Figure 2.14 représente le graphe conceptuel d'une requête permettant de connaître l'âge de Lee au moment où il a été embauché.

La logique descriptive est utilisée par exemple en web sémantique pour la représentation d'ontologies comme dans le langage OWL [36]. Différentes variantes de la logique descriptive permettent plus ou moins d'expressivité. OWL est par exemple disponible

35. ISO 24707 : Common Logic (CL): a framework for a family of logic-based languages
36. Ontology Web Language

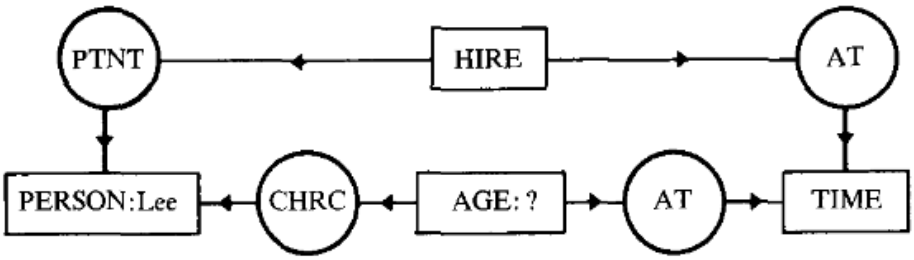


FIGURE 2.14 – Exemple de requête exprimée à l'aide d'un graphe conceptuel

en 3 versions : (1) OWL-Lite utilise la logique de description $\mathcal{SHIFT}$ qui permet d'inférer des résultats dans un temps raisonnable ; (2) OWL-DL utilise la logique de description $\mathcal{SHOIN}(\mathcal{D})$ qui est une version décidable en temps exponentiel dans le pire des cas ; et (3) OWL-Full utilise une logique de description contenant des problèmes indécidables.

2.3.2.3 Problèmes de satisfaction de contraintes

Les problèmes de satisfaction de contraintes (CSP [37]) correspondent à une classe de problèmes où l'on cherche l'état d'un objet en fonction d'un certain nombre de contraintes et de critères. Un CSP est modélisé par un triplet $\langle V, D, C \rangle$ dans lequel V est un ensemble de variables, D est un ensemble de domaines associés aux différentes variables de V et C un ensemble de contraintes sur ces variables. Un CSP représente donc (1) une classe de problèmes, (2) une manière de représenter les connaissances relatives à cette classe de problèmes et (3) un paradigme de programmation déclaratif.

Une fois un problème représenté à l'aide de règles et de contraintes de manière déclarative, celui-ci est passé à un moteur d'inférence qui, à l'aide d'algorithmes d'arc consistance, va essayer de réduire les domaines de chacune des variables en fonction des contraintes. Une fois ces domaines réduits au maximum, un algorithme de parcours et de backtrack va explorer les valeurs restantes afin de trouver les solutions au problème. À chaque choix effectué par l'algorithme de parcours, la propagation des contraintes est relancée. Les moteurs d'inférences implémentent des heuristiques de parcours. La satisfaction de contrainte est un problème NP-complet [Freuder 82] équivalant à un problème de K-coloration de graphe et donc à une recherche d'isomorphisme d'un graphe dans un graphe K-complet [Garey 76].

Les travaux de ce domaine de recherche se focalisent sur la proposition (1) d'algorithmes de propagation de contraintes garantissant leur consistance localement [Mackworth 77, Montanari 74, Gaschnig 74] avec notamment les méthodes d'arc consistance [Bessiere 94, Kumar 92].; (2) d'heuristiques de parcours permettant de trouver une ou toutes les solutions une fois les contraintes évaluées [Haralick 80, Minton 92, Selman 92, Stutzle 05] ; et enfin (3) d'optimisations logicielles [Dubois 95, Schiex 92].

D'autres travaux ont proposé des méthodes de résolution de contraintes distribuées.

37. Constraint Satisfaction Problem

C'est le cas des travaux de Yokoo et Al. [Yokoo 92, Yokoo 94] dans lesquels chaque variable du problème sous contrainte est distribuée à un agent. Un agent contrôle donc une variable du problème ainsi que toutes ses contraintes. Lors de l'exécution du système, chaque agent coopère en s'échangeant des instances partielles du problème qui sont résolues progressivement. La notion de backtrack, qui est fondamentale dans les moteurs de résolution de contraintes classiques peut être facilement étendue au cas d'un système distribué : si un agent arrive à une inconsistance (le domaine de sa variable est vide), il va effectuer une demande de backtrack auprès de l'agent à l'origine de la requête. Cette résolution peut s'effectuer de manière synchrone ou asynchrone. Contrairement à une résolution asynchrone, dans une résolution synchrone, seul un agent est actif à la fois. Dans ce cas, le seul avantage par rapport à une résolution classique est de permettre de garder, à un niveau local à l'agent, les contraintes du système et donc garantir leur confidentialité. Cette méthode de résolution de contraintes distribuée peut être utilisée dans des problèmes modélisés à base de systèmes multi-agents [Yokoo 12]. Petcu et Al. ont aussi proposé DPOP et ODPOP, deux versions d'un autre algorithme de recherche distribué pour de l'optimisation de contraintes [Petcu 05, Petcu 06].

2.3.3 Digression sur les langages de description pour le Web sémantique

Dans la sous-section précédente, nous avons présenté certains formalismes (notamment à base de graphes) pour représenter et raisonner sur les connaissances. Il existe, un ensemble de langages basés chacun de ces formalismes permettant de modéliser certains types de connaissances. Ces différents langages peuvent servir à modéliser soit des applications spécifiques à un domaine ou à une classe de problèmes, soit des applications ou des connaissances génériques. Le premier type de langage est appelé DSML [38]. Ce sont des langages dits dédiés qui ont été spécifiquement conçus pour répondre aux contraintes d'un domaine d'applications. HTML, SQL, VHDL ou encore OWL sont des exemples de langages dédiés respectivement à la description de la structure d'une page web, la description de requêtes dans une base de données relationnelle, la description du fonctionnement d'un composant électronique et la description d'ontologies pour le web. Le second type est appelé GPML [39]. Contrairement au DSML, ces langages généralistes sont utilisables quelque soit le domaine d'application ciblé. On y trouve des langages de balisage comme XML, des langages de modélisation comme UML ou encore des langages de programmation comme Java. Contrairement à HTML qui est un langage de balisage pour décrire la structure d'une page web, XML est un langage de balisage général qui permet de décrire n'importe quel type de données.

Le World Wide Web (ou tout simplement Web) correspond à l'ensemble des documents et des ressources identifiés à une URI [40] et liés entre eux à l'aide de liens hypertextes accessibles sur l'Internet. Le concept du Web a été inventé par Tim Berners-Lee [Berners-Lee 90]. Les principales ressources composant le Web sont des documents appelés pages web, formatés et annotés à l'aide du langage HTML [41]. Ce langage permet

38. Domain Specific Modeling Langage
39. General Purpose Modeling Langage
40. Uniform Ressource Identifier
41. Hypertext Markup Language

de décrire la structure des informations contenues dans la page et d'indiquer des références vers d'autres pages du web. L'avantage principal est donc de pouvoir lier les ressources mises à disposition par d'autres personnes sans les dupliquer. Ces ressources web forment donc un graphe géant dont les nœuds représentent les ressources et les arcs dirigés, les liens entre ces ressources.

Le web sémantique vise à étendre le Web pour prendre en charge des ressources et des données sémantiquement décrites et donc interprétables par les machines [Berners-Lee 01]. Le W3C [42] propose de standardiser certaines technologies qui permettent de structurer sémantiquement les informations contenues dans les pages web. Ces standards propulsés par le W3C sont des formats de données ainsi que des protocoles d'échanges sur le web. Parmi ces langages de description, l'on trouve HTML, RDF [43], RDFS [44] et OWL qui sont détaillés dans la suite de ce paragraphe.

HTML est un modèle pour décrire des pages web, basé sur le langage de description XML. Chaque élément HTML possède une signification sémantique permettant de décrire les éléments d'une page web. La standardisation de l'HTML5 pousse encore plus loin le concept. Néanmoins, ce langage ne permet pas de décrire des connaissances générales.

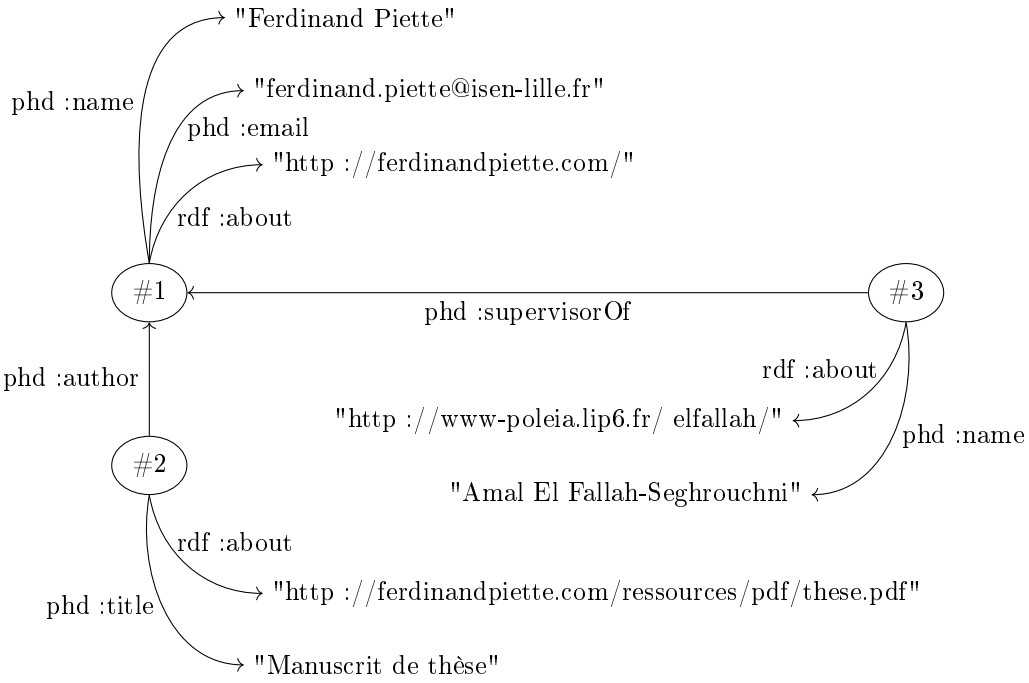


FIGURE 2.15 – Exemple de graphe RDF

RDF est un modèle de graphes qui, à l'aide de triplet, représentent des ressources et des métadonnées [McBride 04]. Un triplet prend la forme *(sujet,prédicat,objet)* où le sujet est une ressource, le prédicat est une relation et l'objet peut être un attribut

42. World Wide Web Consortium
43. Resource Description Framework
44. Ressource Description Framework Schema

caractérisant le sujet ou une autre ressource. Ainsi, une ressource peut être liée à une autre ressource ou à un attribut à l'aide d'une relation. Différents langages permettent d'exprimer des graphes RDF. C'est le cas de RDF/XML qui est un langage basé sur une syntaxe XML. La figure 2.15 représente un exemple d'un ensemble de trois ressources exprimées sous la forme d'un graphe RDF.

— http ://ferdinandpiette.com/ : représente l'identifiant (URI) d'une ressource décrivant une personne. Cette ressource possède deux relations pointant vers des attributs décrivant le nom et l'adresse email de la personne.
— http ://www-poleia.lip6.fr/~elfallah/ : représente l'identifiant d'une ressource décrivant une personne. Cette ressource possède deux relations. La première la liant à un attribut représentant son nom. Et la seconde la liant à une autre ressource dont elle est le superviseur.
— http ://ferdinandpiette.com/ressources/pdf/these.pdf : représente l'identifiant d'une ressource décrivant un document. La relation phd :author permet de lier cette ressource à une autre ressource de type personne correspondant à l'auteur du document.

Le code RDF/XML correspondant à ce graphe est le suivant :

Listing 2.1 – Code RDF/XML de l'exemple représenté à la Figure 2.15

```
<?xml version="1.0"?>
<rdf:RDF
   xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
   xmlns:cd="http://fake.fr/phd#">

   <rdf:Description rdf:about="https://edite-de-paris.fr/public/person/see
       /3163932">
     <phd:name>Ferdinand Piette</phd:name>
     <phd:email>ferdinand.piette@isen-lille.fr</phd:email>
   </rdf:Description>

   <rdf:Description rdf:about="http://ferdinandpiette.com/ressources/pdf/
       these.pdf">
     <phd:author rdf:resource="https://edite-de-paris.fr/public/person/see
         /3163932"/>
    <phd:title>Manuscrit de these</phd:title>
   </rdf:Description>

   <rdf:Description rdf:about="https://edite-de-paris.fr/public/person/see
       /10006318">
     <phd:name>Amal El Fallah-Seghrouchni</phd:name>
     <phd:supervisorOf rdf:resource="https://edite-de-paris.fr/public/person
         /see/3163932"/>
   </rdf:Description>

</rdf:RDF>
```

RDF permet donc de décrire les connaissances sous la forme de graphes. Néanmoins, il n'existe pas de mécanismes permettant de décrire les prédicats, ni les relations entre les prédicats et les ressources. Ces mécanismes sont fournis par un langage de description de vocabulaire RDF : RDFS [Brickley 04]. RDFS est un langage de description permettant de décrire les prédicats RDF ainsi que leurs relations. C'est donc un métamodèle permettant de définir des classes caractérisant des ressources ainsi que des propriétés, caractérisant les prédicats, qui peuvent être hiérarchisées ou instanciées. La

définition de ce métamodèle permet de faire de l'inférence simple sur les modèles RDF. La Figure 2.16 énonce un schéma permettant de modéliser l'exemple de la Figure 2.15. Le schéma en lui-même correspond à la partie du graphe en noir. La partie du graphe en rouge correspond à une partie du graphe RDF de l'exemple précédent. Les liens en bleu relient le graphe RDF au schéma. La partie du graphe en gris correspond à la définition de RDFS. Il est facile de remarquer qu'il est possible de faire de l'inférence

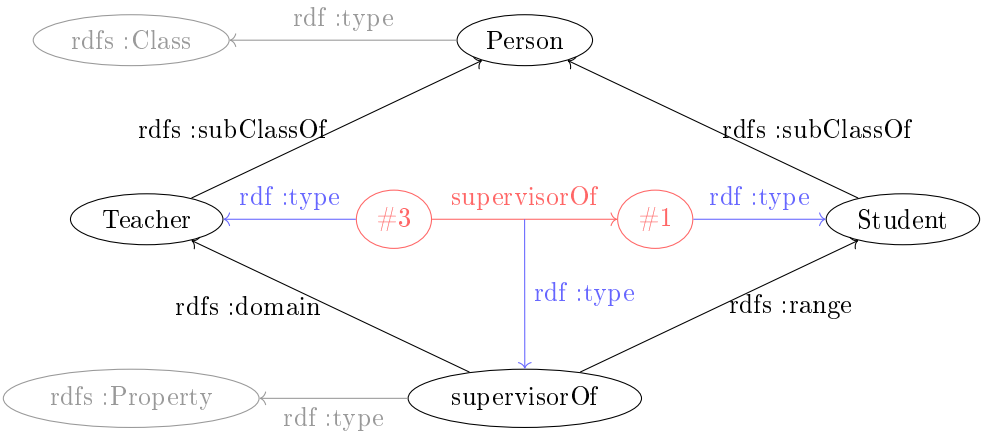


FIGURE 2.16 – Schéma RDF pour l'exemple de la Figure 2.15

grâce au schéma. Ainsi, nous avons spécifié que la propriété « supervisorOf » doit forcement lier une ressource de type Teacher (rdfs :domain) à une ressource de type Student (rdfs :range). Il est ainsi possible de définir des mécanismes qui vérifient dans le modèle RDF que tous les prédicats « supervisorOf » lient bien une ressource de type professeur à une ressource de type étudiant. Néanmoins, ces mécanismes d'inférence sont très limités. En autre, RDF et RDFS ne possèdent pas de moyen d'exprimer la négation ou la disjonction. Son pouvoir expressif est donc réduit ; il est impossible d'exprimer une inconsistance. Pour plus d'expressivité, OWL [McGuinness 04] a été proposé en 2004 comme une recommandation de la W3C.

OWL est un langage de description d'ontologie basé sur la logique descriptive, comme évoquée à la Sous-section 2.3.2.2. En plus de permettre la définition de classes et de propriétés, comme en RDFS, OWL ajoute un ensemble de classes permettant de définir des opérations. Ces opérations peuvent être :

— Des combinaisons de classes : union, intersection, complémentaire
— Des restrictions sur les cardinalités et les valeurs possibles
— Des axiomes sur les classes, les propriétés et les individus : disjonction, équivalence, etc.

L'expressivité d'OWL est donc suffisante pour couvrir tout le formalisme de la logique descriptive. OWL et RDFS permettent conjointement d'exprimer des métamodèles permettant de décrire et de valider des modèles RDF.

2.3.4 Synthèse

Dans cette section, nous avons présenté le domaine de la représentation des connaissances en s'intéressant à différents formalismes de représentation nous permettant, dans

la suite de nos travaux, de décrire de manière efficace l'infrastructure matérielle ainsi que les besoins des applications. Chacun de ces formalismes possède un ensemble de mécanismes permettant de raisonner sur ces connaissances. Ainsi, les formalismes à base de graphes permettent d'utiliser des algorithmes de graph matching permettant de trouver des correspondances d'un graphe source vers un graphe cible. Les formalismes à base de logique permettent d'inférer de nouvelles connaissances à partir d'un ensemble de base. Enfin, le formalisme de représentation à l'aide de contraintes permet de trouver la configuration d'un problème qui respecte un certain nombre de règles et de contraintes déclarées au préalable.

Nous verrons dans les chapitres suivants que ce sont les approches à base de graphes qui ont été retenues. En effet, ces approches ont l'avantage de permettre une modélisation intuitive des entités de l'infrastructure, de leurs propriétés et de leurs relations ainsi qu'une utilisation de mécanismes permettant de rechercher dans ces graphes, des sous-ensembles d'entités respectant un certain nombre de propriétés. Afin de garantir la consistance de ces modèles, les formalismes à base de logique descriptive (et notamment le langage OWL) seront également utilisés. L'ontologie écrite à l'aide d'OWL permettra ainsi d'utiliser des mécanismes d'inférences capables de valider l'écriture des modèles de l'infrastructure disponible et des applications à déployer.

Une approche orientée sur la spécification des contraintes des applications et sur le raisonnement à base de moteurs de résolution de contraintes aurait également été possible. Néanmoins cette approche n'a pas été retenue. Il nous a semblé plus intéressant de décrire l'infrastructure ainsi que les besoins des applications sous la forme de graphes car ceux-ci sont plus intuitifs et plus facile à générer que de modéliser le problème à l'aide de contraintes. De plus, une modélisation orienté contraintes signifie qu'il aurait fallu utiliser un algorithme général de résolution nous laissant peu de choix quant à l'ajout d'heuristiques propres au problème (sauf si nous avions implémenté un moteur de résolution de contraintes spécialement pour notre problème), alors qu'une modélisation orienté graphes nous a permis de développer un algorithme spécifique de graph-matching nous laissant une plus grande liberté quant à son implémentation et aux choix de parcours. Enfin, comme le problème de graph-matching peut être réécrit sous la forme d'un ensemble de contraintes [Larrosa 04], les deux approchent peuvent être considérées comme équivalentes.

2.4 Conclusion

Dans ce chapitre, nous avons dressé un état de l'art en trois parties. La première partie s'est concentrée sur l'étude des applications et des intergiciels permettant la gestion des applications au sein d'environnements ambiants. Un accent a été mis sur l'étude de la confidentialité des informations transitant dans les systèmes distribués. Dans cette partie, nous avons déterminé un certain nombre de bonnes propriétés que doivent avoir les intergiciels de gestion des environnements ambiants. Parmi ces bonnes propriétés, nous retiendrons notamment : la distributivité de la solution, sa robustesse et sa mise à l'échelle ainsi que la dynamicité de son approche. La seconde partie nous a permis d'avoir un aperçu des techniques utilisées pour le déploiement automatique et la configuration d'applications sur des infrastructures composées d'entités de type

ordinateur. Le déploiement de services introduit des techniques de raisonnement sur des modèles pour configurer et/ou déployer automatiquement un ensemble de services composant une application. Le déploiement sur Grille de calcul explore les travaux d'un point de vue scientifique alors que le déploiement dans le Cloud Computing présente des travaux plus pragmatiques et utilisés par les industriels actuellement. Bien qu'il ne soient pas appliqués au déploiement d'applications dans des environnements ambiants, un processus global pour le déploiement (description, raisonnement, exécution) a pu être isolé et pourra être réutilisé par la suite. Enfin, la dernière partie a dressé un tableau des différentes méthodes existantes pour représenter et raisonner sur un ensemble de connaissances.

Dans la prochaine partie du manuscrit, nous présenterons nos différentes contributions. Le Chapitre 3 présentera la manière dont nous avons choisi de modéliser, à l'aide de graphes, l'infrastructure et les applications d'un environnement ambiant. Le Chapitre 4 présentera le modèle mathématique, étendant le concept d'homomorphisme de graphes, permettant de raisonner sur les entités de l'infrastructure à utiliser pour déployer une application et ainsi établir un plan de déploiement. Les algorithmes basés sur ce formalisme seront présentés au Chapitre 5 avec d'abord une version centralisée adaptant un algorithme de graph matching de type Branch and Bound existant, puis sa version décentralisée garantissant la confidentialité des informations relatives à l'infrastructure matérielle. Enfin, le Chapitre 6 présentera une architecture multi-agents fournissant les mécanismes permettant de déployer et de configurer automatiquement des applications distribuées dans un environnement ambiant.

Deuxième partie

Contribution

Chapitre 3

Modélisation des systèmes ambiants

> Les concepts sont comme une grille transparente au travers de laquelle nous voyons l'univers, et qui nous donnent parfois l'illusion que cette grille est réellement cet univers.
>
> *Dune, la génèse - La guerre des machines* - Brian Herbert

Sommaire

Ce travail a pour but de faciliter la gestion et le déploiement d'applications fonctionnant dans des environnements intelligents tels que des habitations, des quartiers ou des villes. L'utilisateur choisit l'application à déployer à partir d'un store d'applications et le système sélectionnera les entités matérielles environnantes à utiliser et à interconnecter. Nous voulons pouvoir choisir automatiquement les entités matérielles disponibles dans l'infrastructure qui serviront au fonctionnement des applications.

L'Internet des Objets vise à fournir une infrastructure énergétique et de communication interopérable alors que l'Intelligence Ambiante se focalise sur la gestion du contexte dans les applications intelligentes. Néanmoins, peu de travaux proposent de raisonner sur le déploiement d'applications sur une infrastructure matérielle. Comme nous l'avons vu au Chapitre 2, ces travaux font généralement abstraction de l'hétérogénéité du matériel et des protocoles pour se concentrer sur le déploiement d'applications sur des réseaux composés de périphériques relativement homogènes.

Dans des systèmes ambiants tels que les smart-homes ou les smart-cities, il est malheureusement utopique de miser sur l'homogénéité du matériel ou des protocoles. En effet, les applications de ces systèmes ont besoin d'utiliser différents protocoles de communication en fonction du critère à privilégier (distance de communication, consommation énergétique, débit désiré, etc.). De plus, les entités présentes dans l'infrastructure matérielle de ces systèmes ne peuvent être réduites à des ordinateurs. Elles peuvent être de type capteur, actionneur, calculateur et avoir des puissances et des fonctionnements divers et variés. Cette hétérogénéité est certes un réel casse-tête pour concevoir des applications génériques, décorrelées de l'infrastructure, mais c'est aussi un atout indéniable pour concevoir des applications performantes en termes de coût de communication ou de consommation énergétique par exemple. Il est donc indispensable de prendre en compte l'hétérogénéité du matériel et des protocoles de communication pour pouvoir gérer les applications dans un environnement intelligent.

La gestion de cette hétérogénéité passe par une modélisation des systèmes ambiants, de l'infrastructure matérielle, des applications déployables ainsi que des applications déployées. Une fois notre système ambiant décrit, nous pourrons ensuite raisonner sur les descriptions de l'infrastructure et des applications afin de trouver quelles entités matérielles utiliser pour faire fonctionner ces applications.

Nous proposons dans ce chapitre de décrire les systèmes ambiants et plus particulièrement les environnements intelligents de type smart-homes. Nous introduisons à la Section 3.1 un scénario conducteur illustrant le déploiement d'une application de portier vidéo au sein d'un logement. Nous présentons ensuite, dans la Section 3.2 un métamodèle permettant de modéliser des environnements intelligents de type smart-home, en termes d'infrastructures et d'applications. La Section 3.3 présentera des exemples d'applications et d'infrastructures modélisés à l'aide de cette ontologie.

3.1 Scénarios

Le scénario que nous allons utiliser dans cette thèse met en valeur à la fois l'aspect dynamique du déploiement d'applications distribuées dans le cadre des Maisons

Intelligentes, ainsi que les problèmes de sécurité sous-jacents.

3.1.1 Le portier vidéo de M. Snow

M. Snow utilise une application de portier vidéo dans son appartement ; lorsqu'une personne sonne à sa porte, l'image enregistrée par la caméra d'entrée est affichée sur un écran localisé près de M. Snow. Celui-ci peut alors discuter avec cette personne et décider de lui ouvrir la porte à distance, ou non.

C'est samedi matin et M. Snow attend un colis qui doit être livré chez lui par un transporteur dans la journée. Après un généreux petit déjeuner, il va faire sa toilette dans la salle de bain. À ce moment, son voisin, M. Den, sonne à la porte. La maison intelligente, consciente que M. Snow se trouve dans la salle de bain, sélectionne le miroir connecté présent dans cette pièce, plutôt que n'importe quel autre écran, afin d'être le support d'affichage de l'image de la caméra d'entrée. M. Snow, ne pouvant pas recevoir son invité, l'informe grâce au microphone présent dans le miroir, qu'il lui rendra visite dans une heure environ. En effet, avant d'aller voir M. Den, M. Snow a quelques courses à faire à deux pâtés de maison de chez lui. Pendant son déplacement, il reçoit une notification sur son smartphone : quelqu'un a de nouveau sonné à la porte. À cause du manque d'information concernant la position de M. Snow, la maison intelligente a décidé d'utiliser le smartphone de celui-ci comme terminal par défaut pour afficher le flux d'images de la caméra. Cette fois, c'est le frère de M. Snow qui est venu lui rapporter une série de CDs audio que ce dernier lui avait prêté la semaine précédente. Après l'avoir informé qu'il n'était pas chez lui, M. Snow déverrouille à distance la porte d'entrée de sorte que son frère puisse lui déposer les CDs sur la table. De retour de ses courses, M. Snow se rend enfin chez son voisin, comme il le fait tous les samedis matin. Au beau milieu de leur conversation, il est notifié qu'une troisième personne a sonné chez lui. Cette fois-ci, c'est un inconnu. M. Snow décide alors d'afficher l'image de cet homme sur la télévision de M. Den pour demander à ce dernier s'il connaît le visiteur. Originellement, M. Snow n'a pas les autorisations pour utiliser les périphériques présents chez son voisin. Mais comme lui et M. Den sont de bons amis, ce dernier lui a autorisé l'accès à sa télévision lorsqu'il se trouve chez lui. L'application du portier vidéo est alors redéployée dynamiquement afin d'utiliser les entités matérielles choisies. Comme ni M. Snow ni son voisin ne connaissent le visiteur, M. Snow décide d'activer le microphone de la caméra d'entrée afin de dialoguer avec cette personne. Après avoir compris qu'il s'agissait en fait du transporteur attendu, M. Snow l'informe qu'il arrive de suite. Après avoir remercié M. Den, il retourne chez lui afin de recevoir le colis en main propre.

3.1.2 Analyse du scénario

Ce sénario illustre la manière dont une application (ici, le portier vidéo) est déployée dynamiquement dans l'environnement tout en considérant le contexte utilisateur. Ce scénario met en valeur trois situations de déploiement. Dans la première, l'application est déployée afin d'utiliser une entité matérielle (ici le miroir de la salle de bain) au sein de la maison de l'utilisateur, en fonction de la localisation de celui-ci. Dans la seconde,

l'application est déployée sur le téléphone de l'utilisateur comme solution par défaut, le système n'arrivant pas à localiser l'utilisateur. Et dans la troisième, l'application est déployée sur l'infrastructure d'un autre utilisateur, en vérifiant au préalable que les autorisations nécessaires ont bien été fournies. La localisation est ici fournie par une autre application qui remonte au système l'emplacement de l'utilisateur, lorsque cela est possible.

La question de la confidentialité des ressources est illustrée par les interactions entre les périphériques de M. Snow et M. Den. En effet, il est indispensable de proposer des mécanismes garantissant que ces ressources ne puissent pas être utilisées n'importe comment ni même connues de n'importe qui.

La question de la dynamicité est illustrée par le fait que les périphériques choisis pour le déploiement de l'application doivent s'adapter au contexte. Si l'utilisateur se trouve chez lui, mais change de pièce, l'application doit être redéployée sur le bon écran se trouvant à côté de l'utilisateur.

3.1.3 Extension du scénario

Le scénario précédent utilise une application de portier vidéo ainsi qu'une application de localisation et permet d'illustrer le problème du déploiement d'applications dans un environnement ambiant. Néanmoins, il nous faut encore illustrer le cas où deux applications sont dépendantes l'une de l'autre. Ou plus exactement lorsqu'une application dépend du déploiement d'une autre application. C'est pourquoi nous avons choisi de présenter une extension au premier scénario dans lequel M. Den peut contrôler sa télévision, soit de manière classique avec sa télécommande, soit à l'aide d'un jeu de carte. Sur chacune des cartes est dessinée une action (un logo d'une chaîne de télévision, une icône pour augmenter ou diminuer le son, etc.). Lorsque M. Den présente une carte à la caméra située au-dessus de sa télévision, cette dernière exécute l'action présentée.

Cette extension met en œuvre deux nouvelles applications : l'application télévision ainsi que l'application de commande de la télévision par jeu de cartes.

L'application télévision va venir récupérer le flux vidéo d'une chaîne de télévision et afficher ce flux sur un écran d'affichage ayant une taille suffisante et se trouvant près de l'utilisateur. Le déploiement de l'application dépend donc de la position de l'utilisateur, tout comme l'application du portier vidéo.

L'application jeu de cartes quant à elle va venir utiliser une caméra proche de l'écran utilisé par l'application télévision afin de reconnaître les cartes que l'utilisateur va venir lui présenter. Le dialogue entre l'application de commande et la télévision n'est pas le point qui nous préoccupe ici. Ce qui nous intéresse est de pouvoir utiliser une caméra qui se trouve à côté de l'écran sur lequel est déployée l'application télévision. Ainsi, si l'utilisateur change de pièce, l'application télévision est redéployée afin d'utiliser un nouvel écran. L'application de commande de la télévision devient donc inconsistante et doit être redéployée afin, si possible, de pouvoir utiliser une caméra plus pertinente. Bien évidemment, dans certains cas, le redéploiement de l'application de contrôle n'est

pas possible. Dans ces cas-là, l'utilisateur ne pourra pas utiliser son jeu de carte pour changer de chaine, mais devra retourner à d'autres moyens plus classiques de contrôle, comme l'utilisation d'une télécommande.

3.1.4 Synthèse

À travers ces scénarios, simples de premier abord, transparaît donc des questions complexes comme la gestion de la dynamicité des environnements, la prise en compte de la confidentialité des ressources des utilisateurs, la prise en compte du contexte dans les mécanismes de raisonnement pour le déploiement d'applications ou encore l'interdépendance d'applications. Ce dernier point, bien que soulevé à la Sous-Section 3.1.3, ne sera pas développé dans le cadre de cette thèse. Nous proposerons néanmoins une modélisation des applications permettant, dans des travaux futurs, de facilement intégrer cette problématique.

3.2 Métamodèles pour les maisons intelligentes

Dans la section précédente, nous avons introduit deux scénarios illustrant le déploiement d'applications dans un environnement ambiant. Afin de trouver des mécanismes permettant de trouver dans l'infrastructure matérielle les entités qui serviront de support aux applications, il est d'abord indispensable de modéliser le système, c'est-à-dire, les entités de l'infrastructure matérielles disponibles, leurs propriétés et leurs relations, ainsi que les applications à déployer et leurs besoins minimums en terme d'entités matérielles pour qu'elles puissent s'exécuter avec une qualité de service minimale. Cette thèse se place dans le cadre des maisons et habitats intelligents. C'est pourquoi nous proposons dans cette section un métamodèle permettant de décrire l'infrastructure et les applications de tels environnements. Ce métamodèle pourra être étendu par la suite pour d'autres types d'environnement.

3.2.1 Choix du formalisme

L'infrastructure matérielle est composée d'entités matérielles disponibles interagissant entre elles et possédant des propriétés. Les applications possèdent différentes fonctionnalités interagissant entre elles et décrivant la structure de celles-ci, ainsi que des besoins en termes d'entités matérielles pour que celles-ci puissent fonctionner avec une certaine qualité de service. L'infrastructure disponible et les applications à déployer peuvent se modéliser aisément à l'aide de graphes dans lesquels les nœuds représentent des entités matérielles ou des fonctionnalités reliées sémantiquement entre elles par des relations. Les mécanismes permettant de raisonner sur de tels modèles sont présentés aux Chapitres 4 et 5.

Le métamodèle présenté ici est une ontologie spécifiant le vocabulaire utilisable ainsi que les contraintes à respecter lors de l'établissement des modèles de l'infrastructure et des applications. Pour l'élaboration de ce métamodèle, nous utiliserons un langage

de représentation d'ontologie, plus particulièrement le langage OWL qui a l'avantage d'offrir une sémantique formelle pour la description de ressources sous la forme de graphes. Comme nous l'avons présenté dans le Chapitre 2, il est donc possible, grâce à une ontologie écrite en OWL de faire de l'inférence sur les modèles et ainsi vérifier leur validité et leur cohérence.

La Figure 3.1 représente le modèle graphique utilisé pour représenter le métamodèle. Les classes (owl :Class) sont représentées par des boites rectangulaires indiquant le nom de la classe ainsi que les attributs des instances. Les relations (owl :ObjectProperty) sont représentées par des cercles. Les instances de relations (rdf :property) sont représentées par des liens labellés par le nom de la relation. Par soucis de lisibilité, les relations owl :subClassOf et rdf :typeOf sont symbolisées respectivement par une flèche indiquant l'héritage et par une flèche indiquant l'instanciation. Enfin, les contraintes entre plusieurs éléments sont indiquées par un trait haché liant les éléments contraints aux types de contraintes.

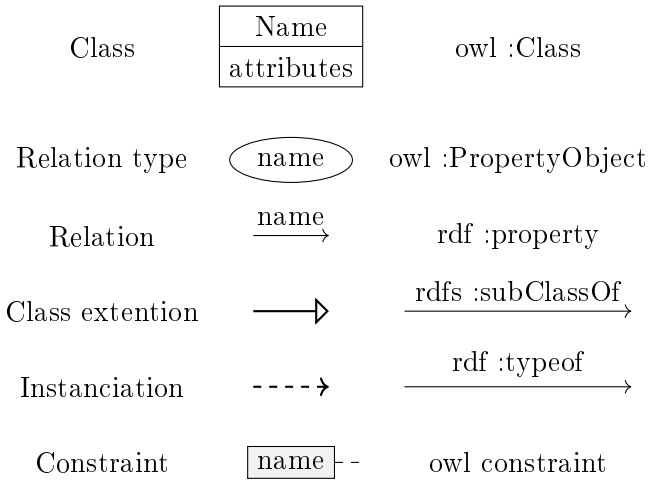


FIGURE 3.1 – Modèle graphique pour le métamodèle OWL

3.2.2 Infrastructure matérielle disponible

L'infrastructure matérielle ainsi que les besoins matériels des applications sont représentés par un graphe bipartite composé de concepts liés entre eux par des relations. Chacun de ces concepts possède un ensemble de propriétés décrivant l'entité de l'infrastructure disponible ou requise. Dans cette sous-section, nous détaillerons les différentes entités possibles qui permettent de décrire l'infrastructure matérielle présente dans une maison, ainsi que les relations possibles pour chacune de ces entités. La Figure 3.2 représente la hiérarchie de classe des quatre types de concepts détaillés dans cette section :

— Physical Entities : représente le concept d'entités physiquement présentes dans l'infrastructure
— Network : représente le concept de réseaux virtuels
— Area : représente une localisation géographique ou un lieu
— Software : représente les entités logicielles

Ces classes de concepts sont toutes disjonctives les unes par rapport aux autres. C'est à dire qu'aucune instance de concept ne peut-être à la fois un réseau et une entité matérielle, ni un logiciel et un lieu, etc.

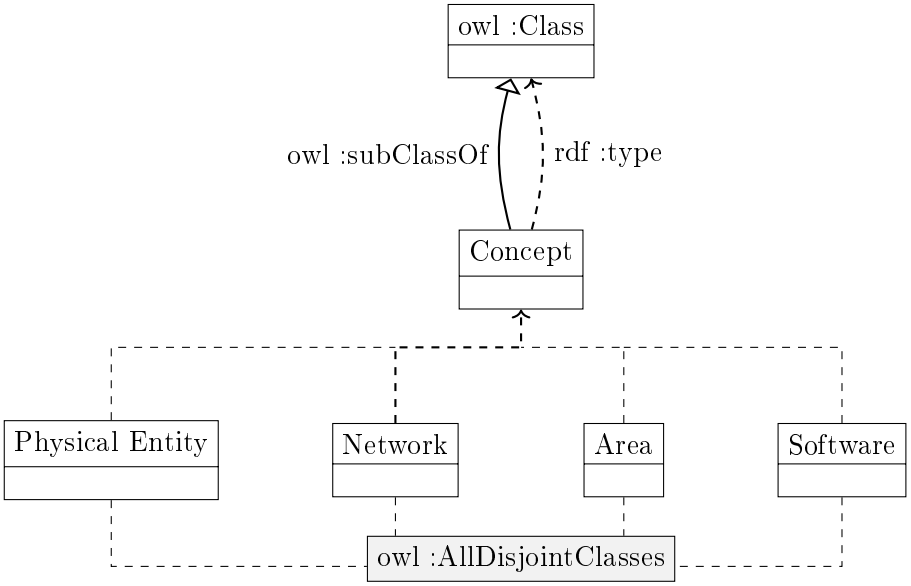


FIGURE 3.2 – Hiérarchie de classe des concepts

3.2.2.1 Les entités matérielles

Les entités physiques présentes au sein d'une habitation peuvent être, soit des entités matérielles, soit des utilisateurs. L'utilisateur est un élément privilégié du système. Il interagit avec l'environnement et influe sur le déploiement des applications. Il est donc important de le représenter dans le graphe de l'infrastructure. Les entités matérielles peuvent être abstraites en quatre catégories de concepts : Sensor, Actuator, Calculator et Inert.

— La catégorie Sensor regroupe les entités matérielles capables de produire des données unitaires ou des flux de données. Elle regroupe les boutons et interrupteurs, les détecteurs de présence, beacons, les caméras, les claviers, souris, pavés tactiles, les microphones, etc.
— La catégorie Actuator au contraire, regroupe les entités matérielles consommant des données ou des flux de données qui permettent d'agir sur l'état physique de cette entité. On y trouve les lumières (lampes, lumières de couleur, bandeaux lumineux, témoins lumineux, etc.), les affichages ou encore, les périphériques audio de type haut-parleur, etc.
— La catégorie Calculator contient des entités capables aussi bien de consommer des données que d'en produire. Elles permettent de traiter, transformer et/ou relayer des données ou des flux de données. Cela regroupe les ordinateurs, micro-ordinateurs, microcontrôleurs, box TVs, box domotiques, box internet, mais aussi les périphériques de communication comme les cartes ethernet, Wifi, etc.
— Enfin, la catégorie Inert regroupe les entités non forcément électroniques qui n'interagissent pas avec les autres entités, mais qui apportent une information

sémantique implicite. Cela peut être un film sans-tain sur une vitre connectée présente dans une salle de bain ou bien une porte sur laquelle est installée une serrure connectée.

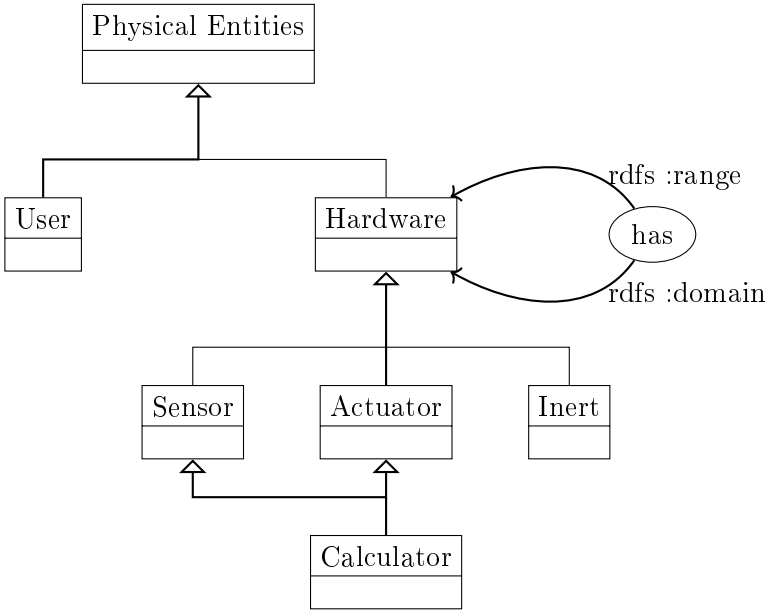


FIGURE 3.3 – Hiérarchie de classes des entités matérielles

Chacune des entités matérielles regroupées dans une de ces quatre catégories possède des propriétés intrinsèques ainsi qu'un ensemble de relations possibles. Toute entité matérielle peut être composée d'autres entités matérielles. La relation *has* représente la composition d'entités matérielles. La figure 3.3 représente la hiérarchie de classes des entités matérielles.

La catégorie Sensor La liste des capteurs intégrables dans une smart-home peut être importante. Il est possible d'avoir des boutons, interrupteurs, détecteurs de présence, beacons, microphones, caméra, souris, touch pad, track ball, touch screen, télécommande, clavier, kinects, leap-motions, capteurs de température, etc.

Pour les besoins de nos scénarios (Section 3.1), nous avons choisi de considérer uniquement les capteurs de type Button, Switch, Camera, Proximity, Beacon et Microphone. Il est bien sûr possible d'étendre facilement cette ontologie afin de rajouter les entités matérielles requises par d'autres scénarios. La Figure 3.4 représente la hiérarchie de classes des différents capteurs considérés.

— Button : Classe générale des boutons qui peut être raffinée en éléments plus spécifiques comme des sonnettes par exemple
— Switch : Classe générale représentant les interrupteurs
— Camera : Classe représentant le concept de caméra. Ses instances possèdent une propriété *framerate* indiquant la plage du nombre d'images par seconde que peut fournir la caméra, une propriété *bitrate* permettant de connaître la bande passante consommée par la caméra, ainsi qu'une propriété *resolution* indiquant la résolution des images du flux vidéo

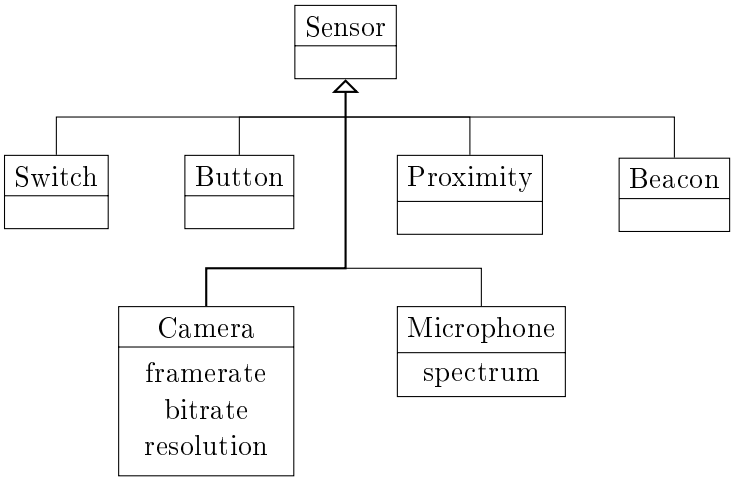


FIGURE 3.4 – Hierarchie de classes des capteurs

— Proximity : Classe représentant un capteur de proximité permettant d'indiquer si une présence humaine est détectée ou non
— Beacon : Classe représentant un type de capteur fournissant une donnée de distance entre un périphérique et le beacon
— Microphone : Classe représentant un microphone. Ses instances possèdent une propriété de type *spectrum* indiquant la plage de fréquences captées par le microphone

La catégorie Actuator Les actionneurs considérés pour le scénario sont de type Relai et Affichage. Le type Relai peut lui-même regrouper d'autres entités comme des lumières ou une serrure de porte commandable. D'autres types d'entités matérielles peuvent être rajoutés à cette catégorie, comme des haut-parleurs, des lumières plus complexes pouvant changer de couleur, etc. La Figure 3.3 représente la hiérarchie de classes des différents actionneurs.

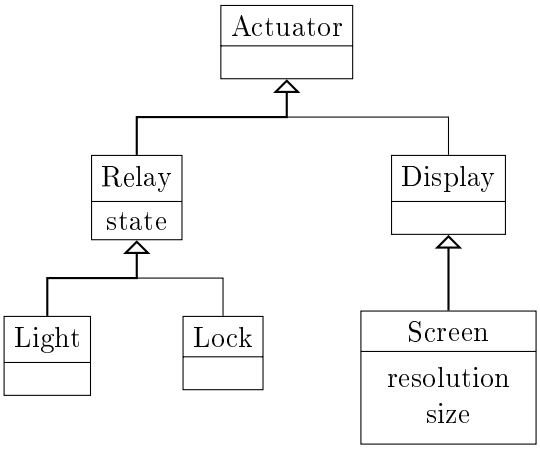


FIGURE 3.5 – Hiérarchie de classes des actionneurs

— Relay : Concept regroupant les actionneurs pouvant être commandés en on/off. Les instances de ce concept possèdent une propriété *state* indiquant l'état de

l'entité
— Light : Une sous-classe de Relay représentant une lumière commandable à distance
— Lock : Une sous-classe de Relay représentant une serrure commandable à distance, permettant par exemple de déverrouiller une porte
— Display : Concept regroupant différents types d'affichage
— Screen : Une sous-classe de Display représentant des écrans. Les instances de ce concept possèdent une propriété *resolution* indiquant l'ensemble des résolutions que peut prendre l'écran, ainsi qu'une propriété *size* indiquant la taille de l'écran

La catégorie Calculator Les calculateurs peuvent être divisés en deux grandes catégories : les ordinateurs et les périphériques de communication. La première regroupe tous les types d'ordinateurs, micro-ordinateurs, box télévisions ou domotiques, microcontrôleurs, etc. La seconde contient des périphériques de communication (IP, EnOcean, Bluetooth, ZWave, Zigbee, LoRa, etc.). La Figure 3.6 représente la hiérarchie de classes des différents calculateurs.

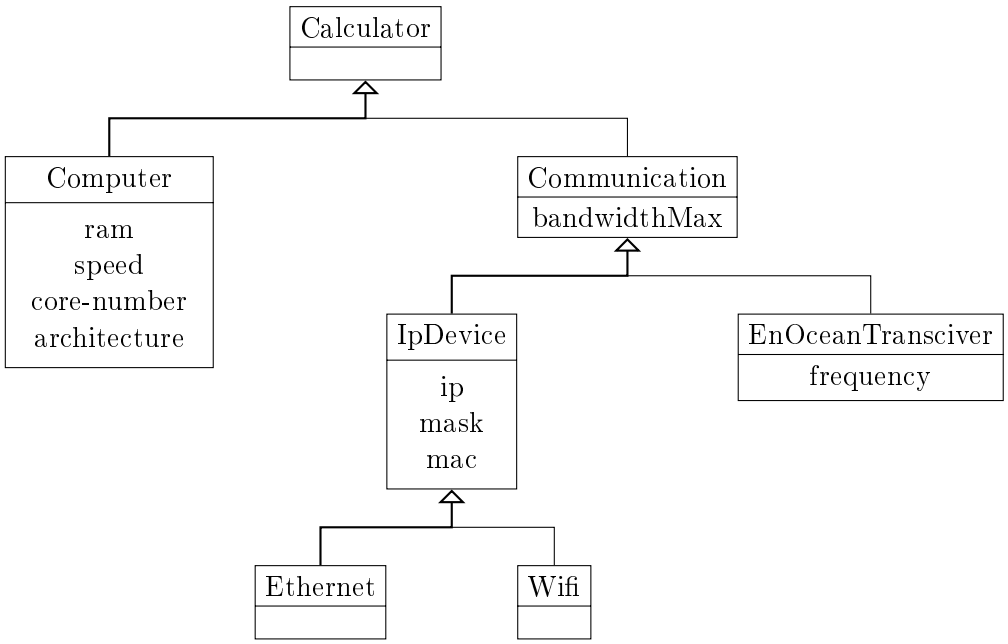


FIGURE 3.6 – Hiérarchie de classes des calculateurs

— Computer : Classe représentant les ordinateurs, micro-ordinateurs, téléphones et tout autre type de périphériques pouvant effectuer un traitement sur des données et pouvant être programmé. Les instances de cette classe possèdent une propriété *ram* indiquant la quantité de mémoire disponible, ainsi qu'une propriété *speed* indiquant la vitesse du processeur ainsi que les propriétés *core-number* et *architecture* indiquant respectivement le nombre de cœurs du processeur ainsi que son architecture (ARM, x86, etc.).
— Communication : Cette classe regroupe tous les périphériques de communication permettant de transférer des données d'un périphérique à un autre. Ses instances possèdent une propriété *bandwidthMax* indiquant la bande passante maximale de l'interface

— IpDevice : Cette sous-classe de Communication regroupe tous les périphériques de communication de type IP. Les instances de cette classe possèdent une propriété *ip* indiquant l'adresse ip du périphérique, une propriété *mask* indiquant le masque réseau ainsi qu'une propriété *mac* indiquant l'adresse mac du périphérique
— EthernetDevice : Est un type d'IpDevice qui est connecté au réseau en filaire
— WifiDevice : Est un type d'IpDevice connecté au réseau par le protocole Wifi
— EnOceanTransceiver : Est un périphérique de communication utilisant le protocole EnOcean, caractérisé par une propriété *frequency* indiquant la fréquence du canal de communication utilisé

La catégorie Inerte Cette catégorie regroupe les entités matérielles qui ne consomment ni ne produisent de données, telles que des portes, des fenêtres, etc. La Figure 3.7 représente la hiérarchie de classes de ces entités. Pour les besoins du scénario, nous avons choisi de conceptualiser uniquement les portes et les films sans-tain.

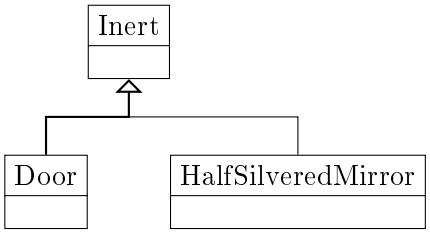


FIGURE 3.7 – Hiérarchie de classes des composants inertes

— Door : Cette classe ne possède pas de propriété particulière. Elle sert à ajouter au graphe d'infrastructure des entités spécifiques de type porte
— HalfSilveredMirror : De même, cette classe indique la présence d'un film sans-tain sur une entité de type écran par exemple

3.2.2.2 Les réseaux de communication

Les entités matérielles présentées dans la section précédente peuvent interagir les unes avec les autres. Ainsi, un micro-ordinateur pourra s'abonner au flux vidéo d'une caméra et afficher certaines informations sur un écran, voire même actionner une lumière si certaines informations contenues dans le flux vidéo sont détectées. Pour cela, il faut donc être capable de décrire la manière dont ces entités peuvent communiquer entre elles au travers de réseaux.

Nous avons donc défini le concept Network. Seules les instances de la sous-classe Communication des entités matérielles peuvent être connectées à une instance de Network grâce à la relation *connectedTo*.

Pour notre application, les réseaux pris en charge sont les réseaux IP et EnOcean. La hiérarchie de classes des réseaux est décrite par la Figure 3.8. Il est bien sûr possible de définir d'autres types de réseaux (Bluetooth, ZWave, etc.) afin d'étendre cette ontologie.

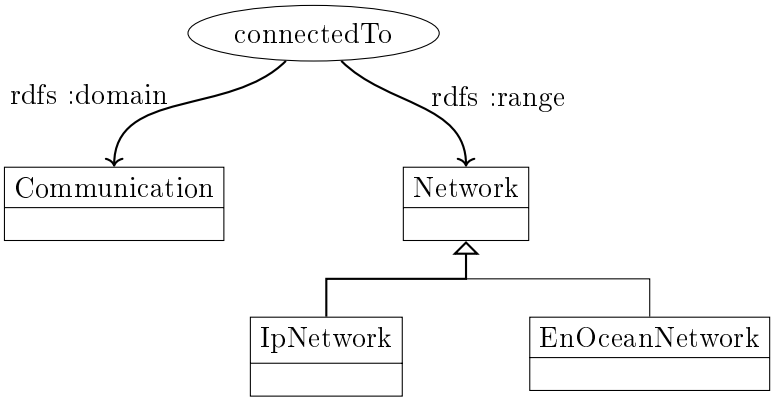


FIGURE 3.8 – Hiérarchie de classes des réseaux

— IpNetwork : Représente un réseau virtuel IP. La propriété *connectedTo* est raffinée et ne peut lier au réseau que des entités de communication de type IpDevice
— EnOceanNetwork : Représente un réseau EnOcean. La propriété *connectedTo* ne peut connecter à ce type de réseau que des entités de type EnOceanTransciver

3.2.2.3 La localisation

Pour les besoins du scénario, il nous faut pouvoir localiser un utilisateur ou une entité au sein d'un logement ou d'un environnement intelligent. C'est pourquoi nous introduisons le concept de Area, ainsi que celui de Room qui sont détaillés à la Figure 3.9. Une entité matérielle ou un utilisateur peut être tagué dans un lieu grâce à la relation *in*.

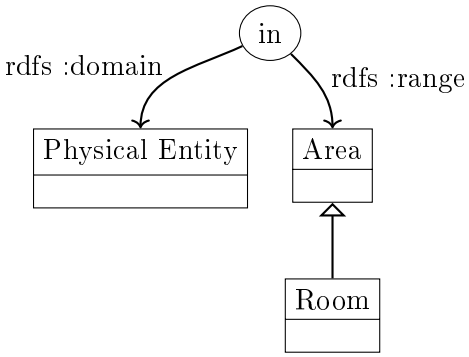


FIGURE 3.9 – Hiérarchie de classes des lieux

— Room : Le scénario n'ayant pas besoin d'un système de localisation précis, nous avons introduit uniquement ici la notion de pièce. Une entité peut ainsi être présente ou non au sein d'une pièce d'un logement

3.2.2.4 Les logiciels

Les instances de la sous-classe Computer des entités matérielles peuvent avoir des logiciels en cours d'exécution. Pour décrire le lien entre entités matérielles et ordina-

teurs, nous avons introduit le concept de *Software*, étendu en *Operating System*, *Web Browser* et *DeploymentArtifact*. La relation *runs* peut être définie entre Computer et OperatingSystem. Un ordinateur exécute un système d'exploitation sur lequel va tourner des logiciels. Cette relation *runs* peut aussi être définie entre deux *Software* quelconques. La Figure 3.10 représente la hiérarchie de classes des concepts logiciels.

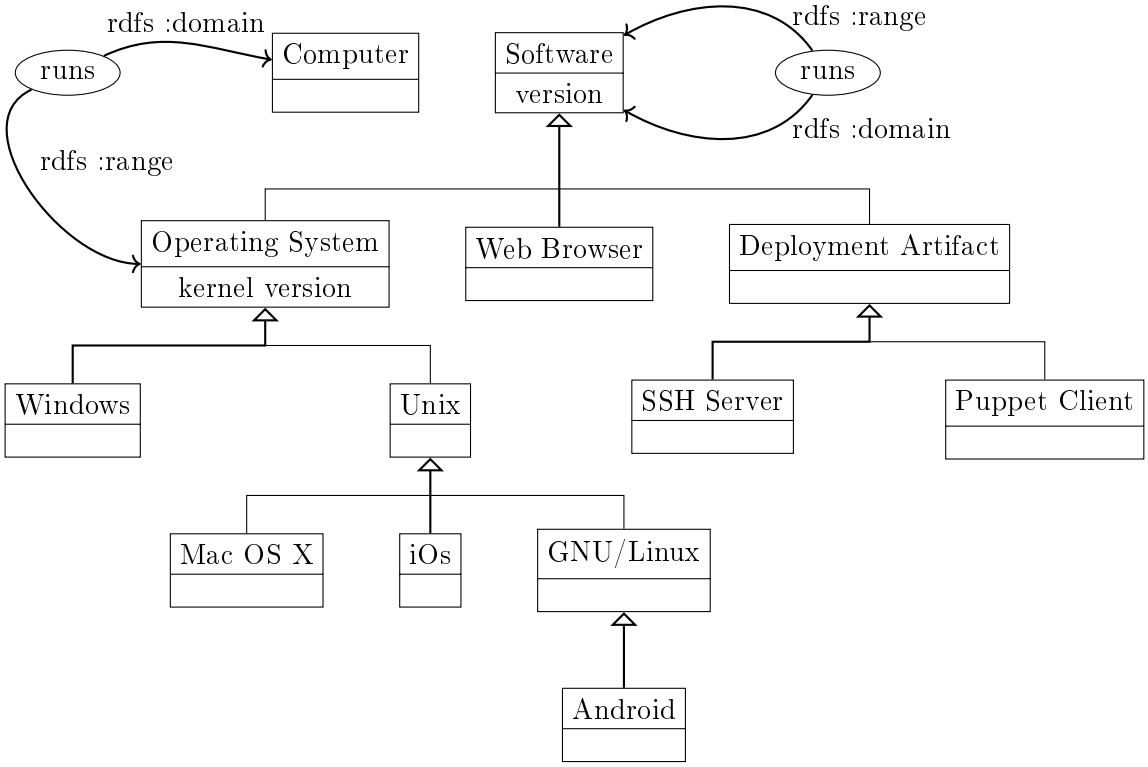


FIGURE 3.10 – Hiérarchie de classes des logiciels

— OperatingSystem : Représente l'ensemble des systèmes d'exploitation dont les instances possèdent une propriété *kernel version* indiquant la version du noyau de l'OS
— Windows : Une sous-classe de système d'exploitation
— Unix : Une sous-classe de système d'exploitation
— MacOSX : Un système d'exploitation de type Unix
— IOS : Un système d'exploitation de type Unix
— GNULinux : Un système d'exploitation de type Unix
— Android : Un système d'exploitation de type GNU/Linux
— Web Browser : La classe des logiciels de type navigateur web dont les instances peuvent exécuter des applications webs
— DeploymentArtifact : La classe des artifacts de déploiement. Cette classe sera utile pour le chapitre 6
— SSHServer : Une sous-classe de Deployment Artifact représentant un serveur Secure Shell
— PuppetClient : Une sous-classe de Deployment Artifact représentant un client de déploiement Puppet

3.2.3 Applications déployables

Les applications déployables peuvent être décomposées en différentes fonctions s'enchaînant les unes aux autres. Ces fonctions sont représentées dans le métamodèle par le concept de fonctionnalité. Pour déployer une fonctionnalité d'une application, une action doit être exécutée. Cette action peut être la configuration d'une entité matérielle ou le déploiement d'une entité logicielle sur un ordinateur. Les entités matérielles configurées ou utilisées pour le déploiement correspondent à des besoins. Ces besoins sont décrits par un graphe d'infrastructure représentant les entités requises pour que la fonctionnalité s'exécute correctement. La relation *executes* renseigne sur le type d'action à exécuter pour déployer la fonctionnalité. La relation *requires* indique l'entité matérielle sur laquelle l'action sera exécutée. Enfin, lors d'une action de déploiement, la relation *deploys* indique l'entité logicielle à déployer.

Ainsi, une application déployable est divisée en deux parties de graphe. La première partie représente les besoins de l'application en terme de matériel pour pouvoir fonctionner. Nous l'appellerons **patron de l'application**. Cette partie du graphe d'application reprend donc l'ontologie détaillée précédemment dans la Sous-section 3.2.2. La seconde partie représente les interactions entre les différentes fonctionnalités de l'application. Ces interactions peuvent nous renseigner sur les données ou les flux de données circulant au sein de l'application, ce qui permettra par la suite de raisonner sur l'emplacement des entités matérielles afin de garantir que certaines données restent au niveau local. D'autres nous permettent de décrire la manière dont les fonctionnalités vont utiliser l'infrastructure matérielle, en détaillant les procédés de déploiement de logiciel, ou la manière de configurer une entité matérielle par exemple. Ce sont ces interactions qui font le lien entre la partie fonctionnalités du graphe d'application et sa partie besoins matériels.

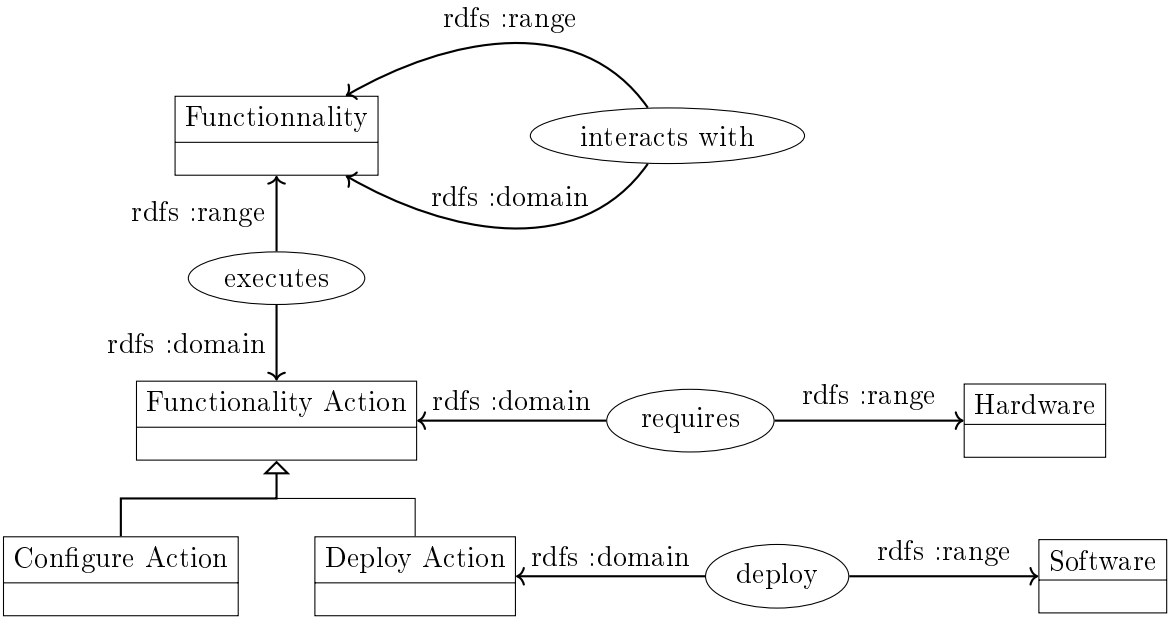


FIGURE 3.11 – Hiérarchie de classes des fonctionnalités

La Figure 3.11 présente la hiérarchie de classes des fonctionnalités. La relation *interacts with* reste abstraite. Celle-ci pourra être spécialisée dans des travaux futurs. La relation *executes* renseigne sur l'action à exécuter afin de déployer la fonctionnalité. L'action en question peut être une action de configuration (*Configure Action)* ou de déploiement (*Deploy Action*). Dans les deux cas, une action interagit avec une entité matérielle de l'infrastructure décrite dans le patron de l'application. La relation *requires* représente ce lien. Enfin, lors d'une action de déploiement, la relation *deploy* indique l'entité logicielle à déployer sur l'entité matérielle.

3.3 Modèles d'infrastructure et d'applications

Dans la section précédente, nous avons présenté un métamodèle introduisant le vocabulaire permettant de modéliser l'infrastructure disponible présente dans un logement, ainsi que les applications déployables. Dans cette section nous utilisons ce métamodèle pour donner des exemples de graphes d'infrastructure et d'applications.

3.3.1 Description de l'infrastructure matérielle existante

Le graphe de l'infrastructure matérielle (ou graphe d'infrastructure) décrit les entités matérielles disponibles, leurs propriétés et leurs interactions. Les instances de concepts représentent les entités matérielles disponibles, les entités logicielles ou virtuelles, alors que les nœuds relation informent sur la sémantique de la relation entre deux instances de concepts. Tout ceci forme un graphe biparti entre instances de concepts et de relations. Chaque nœud du graphe, quel que soit son type, est caractérisé par différentes propriétés. Par exemple, une caméra disponible dans l'infrastructure matérielle est représentée par un nœud qui est une instance du concept Camera et qui possède une propriété framerate décrivant le nombre d'images par seconde que peut fournir la caméra, une propriété bitrate indiquant la bande passante consommée par la caméra, ainsi qu'une propriété *resolution* nous informant sur la taille d'une image générée.

Introduisons à présent un exemple d'un tel graphe. La figure 3.12 est un graphe d'infrastructure simple, composé d'une caméra et d'un ordinateur connectés sur le même réseau. Les rectangles en gras représentent des instances de concepts. Ici, ce sont les nœuds Camera, Computer, Screen, Network, etc. Les rectangles arrondis représentent des nœuds relations qui relient deux nœuds entités (has, runs, isConnectedTo). Le nœud Camera peut capturer 12 ou 24 images par seconde et la bande passante consommée par le flux vidéo sortant est égale au produit du framerate par 512 kilo-octets. Le système d'exploitation de l'ordinateur est de type GNU/Linux, basé sur un noyau en version 3.11. Ce graphe permet donc de décrire les entités matérielles disponibles au sein d'une infrastructure, les relations qu'ils entretiennent, ainsi que leurs propriétés intrinsèques. Il est intéressant de remarquer que la caméra ou l'ordinateur ne sont pas représentés par un unique nœud concept, mais par un ensemble de concepts et de relations formant un sous-graphe. Dans la figure, de tels sous-graphes sont entourés par un rectangle en pointillé. Ceux-ci sont insécables dans le sens où une partie de ce sous-graphe ne per-

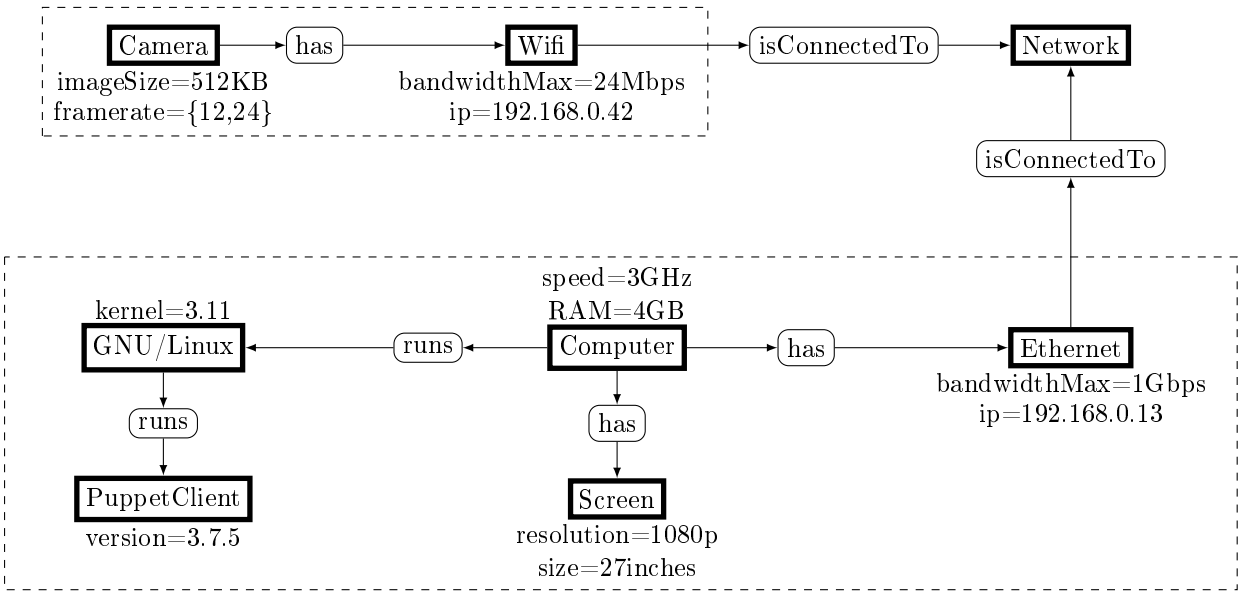


FIGURE 3.12 – Exemple d'un graphe d'infrastructure

met pas de décrire l'entité Caméra ou Ordinateur dans sa globalité. Cette remarque sera particulièrement utile au Chapitre 5 lorsque l'on cherchera à distribuer le graphe d'infrastructure.

3.3.2 Description des applications à déployer

Le graphe d'une application à déployer (ou graphe d'application) décrit la structure ainsi que les besoins d'une application pour que celle-ci puisse fonctionner avec une certaine qualité de service (le patron de l'application). La partie représentant la structure fonctionnelle de l'application est une vue abstraite sur les différentes parties interagissant au sein de celle-ci. Ces différentes parties, appelées fonctionnalités, interagissent aussi bien entre elles, en s'échangeant des données, des flux de données ou des événements, qu'avec l'infrastructure matérielle, soit en utilisant une de ses entités, soit en déployant une entité logicielle dessus. La partie patron de l'application, concernant les besoins matériels de l'application forment un sous-graphe utilisant le même ensemble de concepts et relations que le graphe de l'infrastructure matérielle. Ce patron nous renseigne sur les entités matérielles et les propriétés minimales attendues par l'application pour être déployée et fonctionner normalement. Pour parvenir à ce déploiement, il faut trouver un sous-graphe du graphe d'infrastructure qui corresponde au patron de l'application (sous-graphe des besoins matériels).

Il est possible d'ajouter des contraintes sur les propriétés des nœuds du patron de l'application. Ces contraintes permettent de calculer automatiquement la valeur de certaines propriétés en fonction des propriétés disponibles dans le graphe d'infrastructure. Ce calcul automatique permet ainsi de configurer les entités matérielles et applicatives pendant le processus de déploiement.

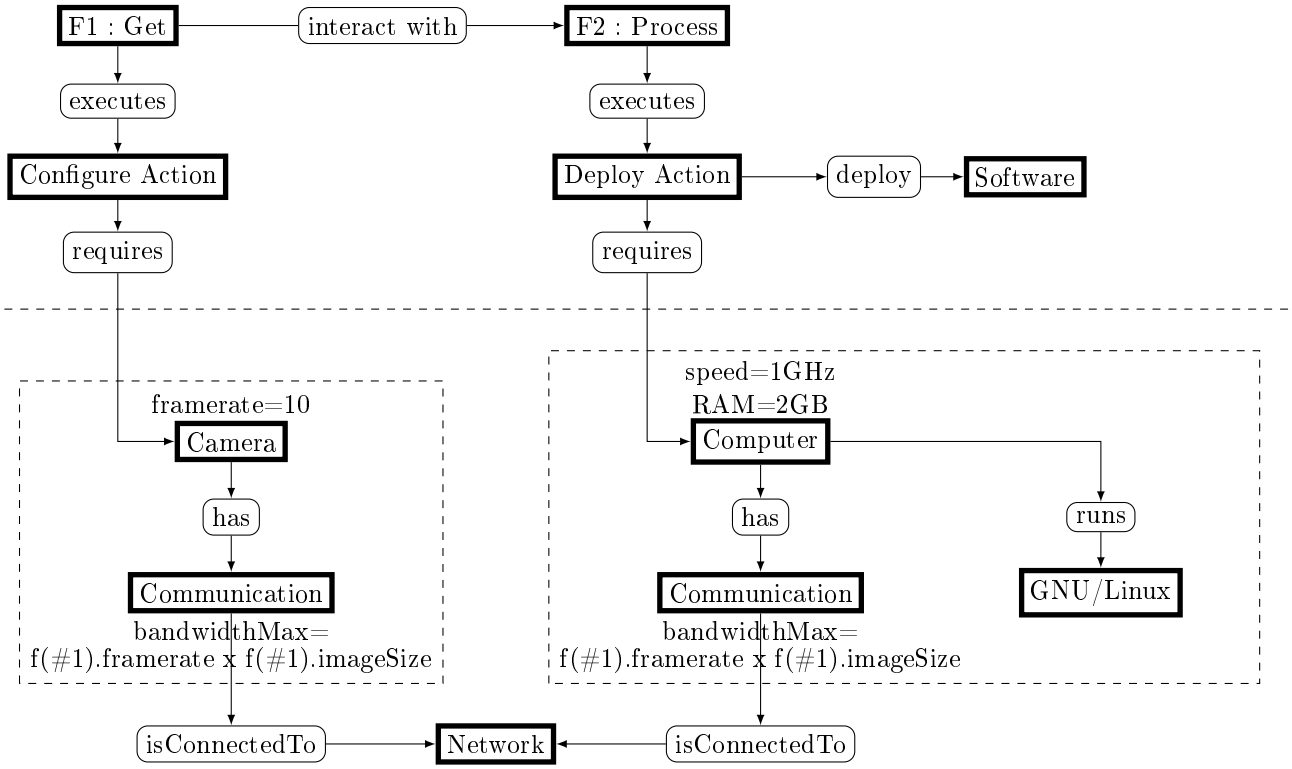


FIGURE 3.13 – Exemple d'un graphe d'application

Considérons que nous voulons déployer une application basique qui capture un flux vidéo d'une caméra et qui effectue un traitement sur ce flux. Cette application est composée de deux fonctionnalités qui s'échangent des données, en l'occurrence, le flux d'images. La première fonctionnalité utilise une caméra disponible dans l'infrastructure matérielle et la seconde doit installer un logiciel sur un ordinateur afin de traiter le flux d'images. La Figure 3.13 nous montre le graphe de cette application. La partie supérieure représente les fonctionnalités de l'application et la manière dont elles seront déployées, alors que sa partie inférieure représente le patron nous informant sur les besoins de ces fonctionnalités en termes d'entités matérielles. Les rectangles en gras représentent les instances de concepts ; les rectangles arrondis, les nœuds relations. Dans cet exemple, l'application a besoin d'une caméra avec un framerate de 10 images par seconde au minimum et l'entité logicielle en charge du traitement du flux d'images doit être déployée sur un ordinateur tournant sous un noyau Linux et possédant au moins 2Go de mémoire RAM. La caméra et l'ordinateur sur lesquels sera déployée l'entité logicielle doivent pouvoir communiquer. La bande passante minimale requise pour faire fonctionner l'application est fonction du bitrate de la caméra. Cette contrainte sur la bande passante permet de configurer automatiquement le framerate de la caméra en fonction de la bande passante disponible lors du déploiement.

3.4 Conclusion

Dans ce chapitre, nous avons proposé un métamodèle basé sur le langage de description d'ontologie OWL à partir duquel il est possible de modéliser l'infrastructure matérielle disponible au sein d'un logement, ainsi que les applications déployables. L'infrastructure matérielle est représentée sous la forme d'un graphe dont les nœuds représentent les entités de l'infrastructure (matériels, utilisateurs, réseaux, localisation, logiciels). Chacun de ces nœuds est caractérisé par un ensemble de propriétés. Ils sont liés entre eux par des relations qui dépendent du type de nœud. Les applications déployables sont considérées comme un ensemble de fonctionnalités interagissant entre elles. Chacune de ces fonctionnalités possède des besoins en termes d'entités matérielles afin que l'application puisse fonctionner avec une certaine qualité de service. Ces besoins sont décrits à l'aide d'un sous-graphe nommé « patron de l'application ». Le formalisme de logique descriptive du langage OWL permet de faire de l'inférence sur les modèles décrits à l'aide de l'ontologie proposée et ainsi vérifier leur consistance.

Le métamodèle présenté est spécifique aux maisons intelligentes. Néanmoins, il est possible d'isoler des concepts communs à tout type d'environnement ambiant. Une perspective pour les futurs travaux serait d'une part, d'étendre la présente ontologie afin de prendre en compte plus d'entités et d'autre part, d'extraire les notions génériques de concept, relation, propriété, fonctionnalité afin de proposer une brique de base à l'élaboration de métamodèles pour les systèmes ambiants.

Les modèles générés à partir du métamodèle permettent de représenter l'hétérogénéité des systèmes ambiants. Dans les chapitres suivants, nous allons voir comment utiliser ces descriptions afin de sélectionner, pour chaque besoin matériel d'une application déployable, l'entité matérielle disponible dans l'infrastructure qui est compatible. Le Chapitre 4 introduit l'homomorphisme de graphe comme un moyen formel de projeter les besoins d'une application sur l'infrastructure. Le Chapitre 5 présente la version centralisée puis décentralisée d'un algorithme de graph matching permettant de trouver une telle projection.

Chapitre 4

Formalisation mathématique du déploiement

Sommaire

Pour déployer une application, nous souhaitons utiliser les technologies et les périphériques hétérogènes existants, développés séparément par différents développeurs, afin d'être le support d'exécution d'applications intelligentes. Nous allons voir dans ce chapitre comment raisonner sur la description de l'environnement introduit au chapitre précédent.

Afin d'être capable de gérer des environnements comme des habitations, des quartiers ou des villes, de manière intelligente, il nous faut à la fois des infrastructures énergétiques et de communication opérationnelles et interopérables, ainsi que des applications intelligentes et adaptables au contexte. Ces deux choses sont déjà opérationnelles, propulsées d'un côté par l'Internet des Objets et de l'autre par l'Intelligence Ambiante. Néanmoins, il manque toujours le lien qui relie ces deux domaines. Il nous manque des mécanismes de projection qui permettent la configuration automatique d'applications et leur déploiement sur l'infrastructure existante. Ces mécanismes de projection doivent pouvoir raisonner sur les propriétés de l'infrastructure matérielle disponible afin de trouver les entités qui seront le support d'exécution des applications intelligentes. Une fois ces entités sélectionnées, il est alors possible de configurer et de déployer effectivement ces applications sur l'infrastructure.

Nous avons présenté dans le Chapitre 3 une modélisation des systèmes ambiants, et plus particulièrement des maisons intelligentes afin d'exprimer l'hétérogénéité de l'infrastructure matérielle et des applications déployables sous la forme de graphes. Dans ce chapitre, ces graphes seront formalisés mathématiquement, ce qui nous permettra de définir une projection comme étant un homomorphisme entre un graphe d'application et un graphe d'infrastructure. Ce mécanisme de projection est défini ici de manière générale, écartant pour le moment toutes considérations liées au domaine d'application (étudiées au Chapitre 3) ou d'ordre algorithmique (Chapitre 5).

4.1 Formalisation mathématique du déploiement

Nous avons choisi de modéliser l'infrastructure matérielle existante et les applications à déployer sous la forme de graphes. Pour déployer des applications sur cette infrastructure, il faut trouver un sous-graphe du graphe d'infrastructure qui supporte les besoins de l'application. Cette projection, respectant les contraintes, nous permet de trouver les entités matérielles à utiliser pour le déploiement de ces applications. Dans cette section, je présente un formalisme mathématique pour le déploiement d'applications. En premier lieu, un ensemble d'opérateurs sera défini afin de pouvoir combiner et comparer les propriétés des nœuds des graphes. Ensuite, nous établissons que le problème de projection des besoins d'une application sur l'infrastructure peut être résolu à l'aide d'homomorphismes de graphe étendus par un axiome sur les propriétés des nœuds. Enfin je montre que ce formalisme est extensible et permet de prendre en compte des situations diverses et intéressantes, comme la configuration automatique de machines virtuelles en fonction des applications à déployer dessus.

4.1.1 Opérations sur les propriétés des nœuds

4.1.1.1 Ensemble de valeurs de propriétés

Chaque nœud du graphe d'infrastructure et du sous-graphe des besoins des graphes d'application est caractérisé par différentes valeurs de propriétés. Je définie P_i, l'ensemble de toutes les valeurs possibles pour une propriété de type i. Dans l'exemple 1, la propriété du type de système d'exploitation (OS) peut avoir pour valeur Windows, Unix, MacOSC, IOs, GNU/Linux ou Android. La propriété de bande passante (bandwidth) peut prendre n'importe quel ensemble de valeurs positives. Et le taux d'images (framerate) d'une caméra peut être une partie de $\mathbb{R}^+$, c'est-à-dire, un ensemble d'intervalles représentant les différents framerates possibles.

Exemple 1

$$
\begin{aligned}
P_{OS} &= \{\text{Windows}, \text{Unix}, \text{MacOSX}, \text{iOS}, \text{GNU/Linux}, \text{Android}, \varnothing, \infty\} \\
P_{Bandwidth} &= \mathbb{R}^+ \cup \{\varnothing, \infty\} \\
P_{Framerate} &= \mathscr{P}(\mathbb{R}^+)
\end{aligned}
$$

À chacun de ces ensembles de valeurs P_i, je définis une loi de composition interne $\vee_i$, ainsi qu'une relation d'ordre partielle $\leqslant_i$ qui permettent respectivement de combiner et de comparer des propriétés d'un même type. Ces paires d'opérateurs permettent de modéliser à la fois des propriétés non consommables (type de système d'exploitation, taux d'images, numéro de version), et des propriétés consommables (bande passante, capacité mémoire). Il est donc possible de rendre une entité non disponible si elle est déjà utilisée par une application ou au contraire, la rendre utilisable par plusieurs applications dans la limite de ses ressources.

4.1.1.2 Loi de composition interne pour la combinaison de propriétés

La loi de composition interne $\vee_i$ sur l'ensemble P_i permet de combiner les valeurs d'une propriété d'un même type i. Si deux nœuds d'un graphe d'application sont projetés sur un même nœud du graphe d'infrastructure, il est possible de combiner les valeurs des propriétés de ces deux nœuds. Le résultat de ces combinaisons nous informe donc sur les valeurs minimales que doivent avoir les propriétés du nœud matériel afin de supporter le déploiement. Dans l'Exemple 2, il est impossible (∞) de déployer sur le même nœud matériel deux nœuds du graphe d'application qui réclament à la fois un système d'exploitation de type Unix et Windows. Par contre, si les systèmes d'exploitation requis sont de types Unix et GNU/Linux, alors le nœud matériel compatible doit être de type GNU/Linux. Si nous voulons déployer deux nœuds applicatifs ayant besoin respectivement de 50Mbps et 20Mbps de bande passante, il nous faut donc trouver un nœud matériel avec au minimum 70Mbps de bande passante disponible. La loi de composition interne pour la bande passante est une simple addition, alors que celle du framerate est une union. Ces opérateurs nous permettent de modéliser aussi bien

des propriétés consommables (bande passante) que des propriétés non consommables (type de système d'exploitation).

Exemple 2

$$Unix \vee_{OS} Unix = Unix$$
$$Unix \vee_{OS} \varnothing = Unix$$
$$Unix \vee_{OS} Windows = \infty$$
$$Unix \vee_{OS} GNU/Linux = GNU/Linux$$
$$50 \vee_{Bandwidth} 20 = 70$$
$$\{5, 10, 15\} \vee_{Framerate} [5, 12] = \{5, 10\}$$

La structure algébrique $(P_i, \vee_i)$ est définie comme un monoïde commutatif absorbant. $\varnothing$ correspond à l'élément neutre, et ∞ est l'élément absorbant. Les propriétés que doivent respecter $(P_i, \vee_i)$ sont donc les suivantes :

— Stabilité : $\forall (x, y) \in P_i^2, x \vee_i y \in P_i$
— Associativité : $\forall (x, y, z) \in P_i^3, (x \vee_i y) \vee_i z = x \vee_i (y \vee_i z)$
— Commutativité : $\forall (x, y) \in P_i^2, x \vee_i y = y \vee_i x$
— Élément neutre $\varnothing$: $\forall x \in P_i, \varnothing \vee_i x = x$
— Élément absorbant ∞ : $\forall x \in P_i, \infty \vee_i x = \infty$

Chaque ensemble P_i possède sa propre loi de composition. Cette loi dépend donc du type de propriété. Elle peut être une simple addition dans le cas de la bande passante, une union pour le taux d'images d'une caméra, une loi d'héritage dans le cas du type de concept, ou n'importe quelle autre loi (opérateur sup, intersection, etc.). Certains opérateurs peuvent avoir d'autres propriétés en plus de celles énoncées ci-dessus. C'est le cas, par exemple, de la structure $(P_{OS}, \vee_{OS})$ pour la propriété caractérisant le type de système d'exploitation qui est idempotent : $\forall x \in P_{OS}, x \vee_{OS} x = x$.

4.1.1.3 Relation d'ordre partielle pour la comparaison de propriétés

La relation d'ordre partielle $\leqslant_i$ sur l'ensemble P_i permet de déterminer si une propriété disponible à droite de l'opérateur possède une valeur suffisante par rapport à la propriété désirée à gauche de l'opérateur. En d'autres termes, cette relation d'ordre signifie "est supporté par". Si la combinaison des valeurs de propriétés de tous les nœuds applicatifs qui sont projetés sur un même nœud matériel respecte la relation d'ordre avec les valeurs de propriétés de ce nœud matériel, alors ce nœud peut supporter le déploiement de cette partie de l'application. Dans l'Exemple 3, une application ayant besoin de 70Mbps de bande passante sur un canal de communication peut utiliser un canal offrant jusqu'à 100Mbps. Ou encore, une application qui a besoin d'une caméra produisant 5 ou 10 images par seconde peut utiliser une caméra présente dans l'infrastructure dont le framerate est de 10 images par seconde. Le cas particulier d'une application ayant besoin d'une machine exécutant un système d'exploitation de type Unix ne peut utiliser qu'une machine tournant sous Unix (ou un système basé sur Unix).

Exemple 3

$$Unix \leqslant_{OS} Unix$$
$$Unix \leqslant_{OS} GNU/Linux$$
$$70 \leqslant_{Bandwidth} 100$$
$$\{5, 10\} \leqslant_{Framerate} \{10\}$$

Cette relation d'ordre $\leqslant_i$ sur l'ensemble P_i doit respecter les propriétés suivantes :
— Réflexivité : $\forall x \in P_i, x \leqslant_i x$
— Anti-symétrie : $\forall (x, y) \in P_i^2, (x \leqslant_i y \text{ et } y \leqslant_i x) \Rightarrow x = y$
— Transitivité : $\forall (x, y, z) \in P_i^3, (x \leqslant_i y \text{ et } y \leqslant_i z) \Rightarrow x \leqslant_i z$

Ici aussi, chaque ensemble de propriétés possède sa propre relation d'ordre qui n'est pas forcement totale. En effet, dans le cas de la relation d'ordre $\leqslant_{OS}$ sur la structure $(P_{OS}, \vee_{OS})$, pour caractériser le type de système d'exploitation, deux valeurs de propriétés ne peuvent pas forcement être mises en relation : $\exists (x, y) \in P_{OS}^2, (x \leqslant_{OS} y) \vee (y \leqslant_{OS} x) \Rightarrow \perp$. Il est aussi intéressant de noter que la relation d'ordre $\leqslant_i$ sur l'ensemble P_i muni d'une loi de composition interne $\vee_i$ est compatible avec cette loi de composition.

$$\forall (x, y, z) \in P_i^3, x \leqslant_i y \Rightarrow x \vee_i z \leqslant_i y \vee_i z$$

Ainsi, la structure algébrique $(P_i, \vee_i)$ forme un monoïde commutatif absorbant ordonnée par $\leqslant_i$.

Le corollaire qui en découle est que $\forall (x, y) \in P_i^2, x \leqslant x \vee_i y$. Ce qui nous permet de dire que le fait de déployer une nouvelle application va contraindre le système et donc, par extension, le fait de désinstaller une application va relâcher certaines contraintes. La désinstallation une application déployée ne va donc jamais remettre en cause le déploiement des autres applications qui ont déjà pu être déployées.

4.1.1.4 Vecteur de propriétés et opérateurs associés

Chaque nœud des graphes possède plusieurs types de propriétés qui le caractérise. Les valeurs de ces propriétés peuvent être représentées par un vecteur de propriétés. Définissons Π, le produit cartésien de tous les ensembles P_i existants.

$$\Pi = P_1 \times ... \times P_n = \prod_{i \in I} P_i$$

Π représente l'ensemble des n-uplets de valeurs possibles des propriétés de la famille I. Un nœud du graphe est donc caractérisé par un de ces n-uplet de Π. Dans l'Exemple 4, le vecteur de propriétés indique un système d'exploitation et une bande passante, mais ne pas posséder de framerate. Toutes ces valeurs de propriétés sont regroupées au sein du même vecteur qui est accroché au nœud en question.

Exemple 4

$$\begin{pmatrix} Unix \\ 50 \\ \varnothing \end{pmatrix} \in \Pi = P_{OS} \times P_{Bandwidth} \times P_{Framerate}$$

On associe à cet ensemble Π une loi de composition interne générale $\circledV$ ainsi qu'une relation d'ordre partielle générale $\circledleq$ définies par :

$$\forall(\pi,\pi') \in \Pi^2, \begin{cases} \pi \circledV \pi' = (p_1 \vee_1 p'_1, \ldots, p_n \vee_n p'_n) \\ \pi \circledleq \pi' \Leftrightarrow (p_1 \leqslant_1 p'_1) \wedge \cdots \wedge (p_n \leqslant_n p'_n) \end{cases}$$

Ces opérateurs nous permettent de manipuler des vecteurs de propriétés entiers plutôt d'une propriété à la fois.

4.1.2 Graphe

L'infrastructure matérielle ainsi que les besoins des applications sont définies par des graphes. Soit le graphe $G = (V, E, \mathcal{P})$:

— V : L'ensemble des sommets
— E : L'ensemble des paires de sommets représentant les arêtes du graphe
— $\mathcal{P}$: Une fonction de V dans Π qui, pour chaque sommet, associe un vecteur de valeurs de propriétés

Chaque sommet de ce graphe est un concept ou une relation qui sont caractérisés par un vecteur de propriétés dans Π. La fonction $\mathcal{P}$ permet de récupérer ce vecteur de valeurs de propriétés. Dans la suite du manuscrit, le mot « graphe » fera référence à ce type de graphe.

4.1.3 Homomorphisme de graphe enrichi

Pour déployer une application sur l'infrastructure matérielle, il faut associer à chaque sommet du graphe représentant le patron de l'application, un sommet compatible dans le graphe de l'infrastructure matérielle. Pour cela, il faut trouver un morphisme du patron d'application vers le graphe de l'infrastructure matérielle en s'assurant que les contraintes et les besoins soient bien respectés.

Soit $\phi : G \longrightarrow H$ une fonction du graphe d'application G (graphe source) vers le graphe de l'infrastructure matérielle H (graphe cible). ϕ est un homomorphisme de graphe enrichi si et seulement si :

$$\phi : V_G \longrightarrow V_H \begin{cases} \forall(u,v) \in E_G, (\phi(u), \phi(v)) \in E_H \\ \forall y \in V_H, \bigcircledV_{x \in \phi^{-1}(y)} \mathcal{P}_G(x) \circledleq \mathcal{P}_H(y) \end{cases}$$

Pour chaque sommet du graphe représentant le patron de l'application, ϕ associe un sommet du graphe de l'infrastructure matérielle où : les arêtes entre les sommets sont respectées ; et la combinaison des vecteurs de valeurs de propriétés des sommets du graphe d'application qui sont projetés sur le même sommet de l'infrastructure matérielle respecte la relation d'ordre partiel avec les valeurs des propriétés de ce sommet image.

L'application inverse ϕ^{-1} associe à chaque sommet du graphe de l'infrastructure, un ensemble de sommets (potentiellement vide) du sous-graphe des besoins du graphe

d'application. Cet ensemble correspond aux sommets du graphe d'application qui sont projetés sur un sommet du graphe de l'infrastructure matérielle.

S'il existe un homomorphisme entre un graphe G et un graphe H, on dit que G se projette dans H. Ce morphisme ϕ représente donc une **projection** de l'application G sur l'infrastructure H.

L'ensemble des morphismes entre ces deux graphes, noté $Hom\,(G,H)$, représente toutes les solutions possibles au problème de projection de l'application sur l'infrastructure matérielle. Le nombre d'homomorphismes maximal possible varie de manière exponentielle avec le nombre de sommets du graphe d'application G :

$$card\,(Hom\,(G,H)) = hom(G,H) \leq |V\,(H)|^{|V(G)|}$$

Trouver une solution au problème général de l'homomorphisme de graphe est NP-Complet [Conte 04]. Dans notre cas, nous avons enrichi le problème général d'un axiome sur les propriétés des sommets des graphes. Si l'on considère que $\forall x \in V\,(G)\,, \mathcal{P}_G(x) = \varnothing$, alors on se ramène au cas d'un homomorphisme de graphe orienté classique. Comme le respect d'une propriété d'un sommet ne peut que contraindre le problème, comme nous l'avons vu dans la sous-section 4.1.1.3, nous pouvons donc affirmer que notre problème d'homomorphisme enrichi reste donc un problème NP-Complet.

4.2 Extension du formalisme

Notre formalisation mathématique du déploiement d'application se modélise très bien à l'aide de la théorie des catégories. En effet, nous pouvons définir une catégorie $\mathcal{C}$ dans laquelle les objets sont les graphes d'application et d'infrastructure et les flèches sont les homomorphismes enrichis qui permettent de projeter un graphe source sur un graphe cible. Le fait d'exprimer notre formalisme à l'aide de la théorie des catégories permet de mettre l'accent sur l'étude des morphismes entre deux objets.

4.2.1 Composition d'homomorphismes

Cette formalisation nous permet de trouver un moyen de déployer des applications sur une infrastructure matérielle hétérogène, mais elle nous permet aussi de prendre en compte différentes situations intéressantes. Comme les besoins des applications ainsi que l'infrastructure matérielle sont modélisés de la même manière, il est possible d'appliquer le processus de déploiement d'une application à une autre application. Ceci peut être nécessaire dans le cas où il faudrait déployer des applications complexes qui réclament de faire des hypothèses, afin de réduire le graphe de l'application à quelque chose de moins complexe. Ceci est rendu possible par le fait que la composition de deux homomorphismes est aussi un homomorphisme. Soit G, H et K trois graphes et ϕ et ψ, deux homomorphismes, $\phi : G \longrightarrow H$ et $\psi : H \longrightarrow K$.

$$\forall y \in H, \bigvee_{x \in \phi^{-1}(y)} \mathcal{P}_G(x) \circledleq \mathcal{P}_H(y)$$

$$\forall z \in K, \bigvee_{y \in \psi^{-1}(z)} \mathcal{P}_H(y) \circledleq \mathcal{P}_K(z)$$

Nous pouvons écrire :

$$\bigvee_{x \in (\psi \circ \phi)^{-1}(z)} \mathcal{P}_G(x) \circledleq \bigvee_{y \in \psi^{-1}(z)} \mathcal{P}_H(y) \circledleq \mathcal{P}_K(z)$$

Si la nouvelle application, moins complexe, peut-être déployée, alors il est assuré que l'application originale peut aussi être déployée de la même manière.

4.2.2 Fonction de concrétisation

Nous pouvons aussi imaginer déployer une application non pas sur une entité matérielle réelle, mais sur une machine virtuelle et ensuite installer cette machine virtuelle sur une machine réelle. En plus de la propriété de composition des morphismes, nous définissons une fonction $\phi_* : H \longrightarrow \Pi$, appelée fonction de concrétisation, associée à l'homomorphisme $\phi : G \longrightarrow H$. Pour chaque nœud de l'infrastructure matérielle H, la fonction ϕ_* associe le vecteur de valeurs de propriété minimum auquel ce nœud peut être réduit afin de supporter le déploiement des applications par l'homomorphisme ϕ.

$$\forall y \in H, \phi_*(y) = \bigvee_{x \in \phi^{-1}(y)} \mathcal{P}_G(x)$$

$$\forall y \in H, \phi_*(y) \circledleq \mathcal{P}_H(y)$$

Cette fonction de concrétisation ϕ_* permet de trouver la configuration minimale de la machine virtuelle qui permet de supporter le déploiement de l'application.

4.3 Conclusion

Les systèmes ambiants sont caractérisés par une forte hétérogénéité matérielle et logicielle. Cette hétérogénéité augmente considérablement la complexité du déploiement automatique de nouvelles applications. Afin de faciliter ce type de déploiement, nous avons considéré que les entités matérielles de l'infrastructure ainsi que les entités logicielles des applications devaient être décorrélées. Ceci permet d'être plus flexible sur l'ajout et le retrait d'entités matérielles ainsi que pour le déploiement de nouvelles applications sur une infrastructure existante.

Dans ce chapitre, j'ai présenté une formalisation mathématique à base d'homomorphismes enrichis permettant de projeter un ensemble d'applications sur une infrastructure matérielle. Cette formalisation décrit une catégorie (au sens de la théorie des catégories) où les objets du système sont modélisés par des graphes dont la nature (matérielle, logicielle ou virtuelle) n'a pas d'importance et où les morphismes modélisent le déploiement. Il permet aussi la configuration des entités matérielles afin de satisfaire les besoins des applications à déployer. Les propriétés matérielles prises en charge incluent des propriétés non consommables, comme le type de système d'exploitation, aussi bien

que des propriétés consommables, comme la bande passante réseau. Lorsque les besoins d'une application sont projetés sur les éléments du graphe d'infrastructure, certaines propriétés peuvent être automatiquement configurées afin de respecter ces exigences. Le déploiement ou le retrait d'une application peut-être fait dynamiquement si l'on met à jour les propriétés des nœuds du graphe d'infrastructure en termes de ressources utilisées par les applications déjà déployées.

Je présente au Chapitre 5 un algorithme permettant de trouver un tel homomorphisme entre deux graphes. Cet algorithme permet de trouver une projection des besoins d'une application à déployer sur une infrastructure matérielle disponible. Le résultat de cette projection pourra ensuite être utilisé afin de déployer concrètement l'application sur l'infrastructure.

Chapitre 5

Génération de plans de déploiement

Sommaire

Dans le Chapitre 4, nous avons formalisé le problème de projection pour le déploiement d'applications sur une infrastructure matérielle existante pour les systèmes ambiants comme un problème d'homomorphisme de graphe enrichi d'un axiome sur les propriétés des nœuds. Ceci nous permet de réutiliser un des algorithmes de graph matching existants et bien étudiés par la littérature (Chapitre 2). Néanmoins, la plupart de ces algorithmes sont centralisés et ne peuvent répondre entièrement à notre problème. En effet, nous cherchons à déployer des applications dans un environnement intelligent comme un logement, un immeuble, un quartier ou une ville. Dans de tels environnements, l'infrastructure matérielle doit être gérée de manière distribuée afin d'assurer aussi bien la mise à l'échelle du processus de déploiement, que la confidentialité des ressources. C'est pourquoi, nous présentons dans ce chapitre une version distribuée d'un algorithme de graph-matching qui permet de résoudre le problème de la projection d'application sur une infrastructure matérielle tout en garantissant la confidentialité du graphe de l'infrastructure. La Section 5.1 propose une modification d'un algorithme branch and bound centralisé pour trouver les homomorphismes enrichis entre deux graphes. Nous introduirons notamment des itérateurs encodant la manière de parcourir les graphes et de sélectionner les nœuds à faire correspondre. Ainsi, l'algorithme sera générique et seul les itérateurs seront à recoder si nous voulons changer la manière de parcourir les graphes. La Section 5.2 étend cet algorithme afin de permettre la génération de projections partielles qui pourront être complétées au fur et à mesure. Enfin, la Section 5.3 introduit une version distribuée de cet algorithme, qui raisonne sur un ensemble de sous-graphes d'infrastructure tout en garantissant leur confidentialité. Dans cette version, l'algorithme présenté à la Section 5.3.2 est utilisé à deux niveaux : au niveau global, les itérateurs sont re-spécifiés pour parcourir le graphe des sous-graphes ; et au niveau des sous-graphes, la première version de l'algorithme est appliquée afin de faire correspondre un sous-graphe d'une application avec le sous-graphe de l'infrastructure ciblée.

5.1 Algorithme Branch and Bound modifié

Pour déployer dynamiquement une application sur une infrastructure qui est elle aussi dynamique, il nous faut un algorithme qui trouve un homomorphisme entre le patron de l'application (le sous-graphe des besoins matériels du graphe d'application) et le graphe de l'infrastructure matérielle disponible. Il nous faut pouvoir, à l'aide de cet algorithme, trouver tous les homomorphismes possibles, les construire étape par étape, pouvoir proposer à tout moment des solutions partielles et autoriser l'implémentation d'heuristiques pour guider la construction de ces solutions. De cette manière, il sera plus aisé de distribuer par la suite la résolution de l'algorithme. Trouver une solution optimale ne nous intéresse pas. Dans notre cas l'optimalité n'est pas définie et trouver une solution est suffisant. Cette solution garantit déjà le respect des contraintes et des propriétés attendues de l'application. Les algorithmes de résolution par contraintes pour le problème d'homomorphisme sont souvent des processus lourds et ne répondent pas à nos attentes. Nous privilégions dans notre cas des algorithmes de Branch and Bound ou de descente du gradient. Je présente dans cette sous-section l'adaptation d'un algorithme Branch and Bound classique, comme celui présenté par [Chein 08], en intégrant l'axiome sur les propriétés des nœuds de notre homomorphisme de graphe

enrichi, présenté dans la Section 4.1.3.

5.1.1 Détails de l'algorithme

Cet algorithme prend en entrée G, le patron de l'application (sous-graphe des besoins matériels du graphe de l'application) à déployer, ainsi que H, le graphe de l'infrastructure matérielle représentant l'environnement intelligent qui devra accueillir cette application. L'algorithme explore en profondeur le graphe G et essaye d'assigner successivement un nœud compatible de H en suivant les liens d'adjacence. Si l'algorithme se retrouve dans une impasse sans solution, celui-ci backtrack alors jusqu'à ce qu'un autre nœud compatible soit trouvé dans le graphe d'infrastructure. Durant l'exécution de l'algorithme, les nœuds explorés dans le graphe d'application G, ainsi que leurs images dans le graphe de l'infrastructure H représentent une solution partielle de l'homomorphisme. L'homomorphisme de graphes est trouvé lorsque tous les nœuds du graphe d'application ont été explorés et possèdent une image dans le graphe de l'infrastructure. Pour trouver les autres homomorphismes, il suffit de garder en mémoire la solution courante et de continuer l'algorithme en backtrackant afin d'explorer les autres branches.

5.1.1.1 Initialisation de l'algorithme

Algorithm 1: initProjection

input : *applicationGraph*, $infrastructureGraph$, $getFirstNodeIt$, $getNeighboutsIt$, $getCompatibleInfrastructureNodesIt$

output: An empty projection structure

begin

```
projection = new Structure();
projection.applicationGraph = applicationGraph;
projection.infrastructureGraph = infrastructureGraph;
projection.firstNodeIt = getFirstNodeIt(applicationGraph);
projection.getNeighboutsIt = getNeighboutsIt;
projection.getCompatibleInfrastructureNodesIt =
 getCompatibleInfrastructureNodesIt;
projection.exploration = new Stack();
projection.solution = new Stack();
return projection;
```

L'Algorithme 1 permet d'initialiser une structure de données contenant notamment deux piles de parcours. La première pile permet d'explorer le graphe d'application en profondeur afin de récupérer les nœuds qui n'ont pas encore été explorés. La seconde pile quant à elle associe pour chaque nœud parcouru dans le graphe d'application un nœud compatible du graphe d'infrastructure et construit ainsi une projection du graphe d'application vers le graphe d'infrastructure. Les fonctions $getFirstNodeIterator$, $getNeighboursIt$ et $getCompatibleInfrastructureNodesIt$ permettent de rendre l'algorithme de graph-matching générique. Celles-ci encodent respectivement la manière dont le premier nœud

du graphe d'application est choisi ; la manière de parcourir le graphe d'application ; et les mécanismes permettant de trouver un nœud compatible dans le graphe d'infrastructure. Chacune des fonctions retourne un itérateur sur les nœuds des graphes. Le fonctionnement de ces fonctions sera détaillé dans la prochaine Sous-section 5.1.2.

5.1.1.2 Exécution de l'algorithme

Algorithm 2: findNextProjection

```
input  : applicationGraph, infrastructureGraph, projection
output: An homomorphism from applicationGraph to infrastructureGraph
begin
    repeat
        projection = step(projection);
    until isProjectionFound(applicationGraph,projection) or
     projection.exploration.isEmpty();
    return projection;
```

L'Algorithme 2 est le point d'entrer de l'algorithme de matching. Il a été implémenté itérativement afin de permettre une exécution pas à pas. Celui-ci met à jour l'état de la structure de données *projection* en s'arrêtant lorsqu'une nouvelle projection est trouvée. La structure de données *projection* a dû être préalablement initialisée grâce à la fonction *initProjection* lors du premier appel à la fonction *findNextProjection*.

Algorithm 3: isProjectionFound

```
input  : applicationGraph, projection
output: true if an homomorphism is found
begin
    foreach node of applicationGraph do
        if undefined == getImage(projection, node) then
            return false;
    return true;
```

L'Algorithme 3 vérifie que l'état courant de la structure de données *projection* possède une image pour chaque nœud du graphe de l'application. Si c'est le cas, alors une projection est trouvée et la fonction renvoie *true*.

Algorithm 4: getImage

```
input  : projection, applicationNode
output: The image of the applicationNode if exists - undefined otherwise
begin
    foreach element of projection.solution do
        if applicationNode == element.applicationNode then
            return element.infrastructureNode;
    return undefined;
```

L'Algorithme 4 parcourt la pile des solutions de la structure de données *projection* et renvoie le nœud infrastructure associé au nœud application passé en paramètre. Si une telle association n'existe pas encore, la fonction renvoie alors *undefined*.

Algorithm 5: getFibers

```
input  : projection, infrastructureNode
output: The fibers of the infrastructureNode if exists - an empty array
        otherwise
begin
    fibers = [];
    foreach element of projection.solution do
        if infrastructureNode == element.infrastructureNode then
            fibers.push(element.applicationNode);
    return fibers;
```

Sur le même principe que l'Algorithme 4, l'Algorithme 5 retourne l'ensemble des antécédents d'un nœud infrastructure ; c'est à dire, l'ensemble des nœuds du graphe d'application qui sont actuellement associés au nœud infrastructure.

Algorithm 6: step

```
input  : projection
output: An updated version of projection
begin
    projection = findNextUnmatchedNode(projection);
    if !projection.solution.isEmpty() and undefined !=
     projection.solution.last().infrastructureNode then
        projection = matchNextUnmatchedNode(projection);
    return projection;
```

L'Algorithme 6 permet une exécution pas à pas. À chaque itération, la pile d'exploration est mise à jour d'une étape et un nouveau nœud non exploré du graphe d'application est associé à un nœud compatible du graphe d'infrastructure. S'il n'y a plus de projection possible, les piles d'exploration et de solution sont vides.

5.1.1.3 Parcours des piles d'exploration et des solutions

Algorithm 7: findNextUnmatchedNode

```
input  : projection
output: An updated version of projection
begin
    if projection.exploration.isEmpty() then
        if projection.solution.isEmpty() then
            applicationNode = projection.firstNodeIt.next();
        else
            projection = invalidateLastMatch(projection);
    else
        repeat
            applicationNode = projection.exploration.last().neighboursIt.next();
            infrastructureNode = getImage(projection, applicationNode);
            if undefined == applicationNode then
                projection.exploration.pop();
            else if undefined != infrastructureNode then
                previousInfrastructureNode = getImage(projection,
                 projection.exploration.last().applicationNode);
                if projec-
                 tion.infrastructureGraph.hasLink(previousInfrastructureNode,
                 infrastructureNode) then
                    applicationNode = undefined;
                else
                    projection = invalidateLastMatch(projection);
        until undefined == applicationNode;
    if undefined != applicationNode then
        projection = addNodeToMatch(projection, applicationNode);
        projection = addNodeToExplore(projection, applicationNode);
    return projection;
```

L'Algorithme 7 continue le parcours du graphe d'application en profondeur et récupère un nœud non encore exploré. Si les deux piles de parcours sont vides, la fonction $findNextUnmatchedNode$ initialise la pile d'exploration avec un premier nœud récupéré grâce à l'itérateur $firstNodeIt$. Si la pile d'exploration est vide, mais pas celle des solutions, cela signifie que la structure de données *projection* possède une projection complète. Pour trouver la suivante, l'algorithme invalide donc le dernier élément de la pile des solutions. Si les deux piles sont non vides, l'algorithme continue à explorer le graphe d'application. S'il n'y a plus de nœuds à explorer dans la branche en cours, l'algorithme backtrack sur la pile d'exploration afin de trouver une autre branche du graphe d'application qui n'a pas encore été explorée. Si le nouveau nœud choisi possède déjà une solution dans la pile des solutions, l'algorithme va vérifier que les liens dans le graphe d'infrastructure sont corrects. Si ça ne l'est pas, le dernier élément dans la pile des solutions est invalidé, sinon, on continue à explorer le graphe d'application.

Enfin, si le nœud application choisi n'a pas encore d'image, celui-ci est ajouté à la pile d'exploration et des solutions.

Algorithm 8: addNodeToMatch

```
input  : projection, applicationNode
output: An updated version of projection
begin
    solutionElement = new StackElement();
    solutionElement.applicationNode = applicationNode;
    solutionElement.parentNode = this.exploration.last().applicationNode;
    solutionElement.compatibleInfrastructureNodesIt =
     projection.getCompatibleInfrastructureNodesIt(projection,
     solutionElement.parentNode, applicationNode);
    projection.solution.push(solutionElement);
    return projection;
```

Algorithm 9: addNodeToExplore

```
input  : projection, applicationNode
output: An updated version of projection
begin
    explorationElement = new StackElement();
    explorationElement.applicationNode = applicationNode;
    explorationElement.neighboursIt =
     projection.getNeighboursIt(projection.applicationGraph, applicationNode);
    projection.exploration.push(explorationElement);
    return projection;
```

Les Algorithmes 8 et 9 ajoutent un élément respectivement à la pile des solutions et à la pile d'exploration. Des itérateurs de parcours des nœuds compatibles dans le graphe d'application et de parcours des nœuds voisins dans le graphe d'application sont créés grâce aux fonctions $getCompatibleInfrastructureNodesIt$ et $getNeighboursIt$ et attachés aux nouveaux éléments des piles. Le fonctionnement de ces fonctions sera

détaillé dans la sous-section suivante.

Algorithm 10: matchNextUnmatchedNode

```
input  : projection
output: An updated version of projection
begin
    repeat
        infrastructureNode =
          projection.solution.last().compatibleInfrastructureNodesIt.next();
        if undefined != infrastructureNode then
            projection.solution.last().infrastructureNode = infrastructureNode;
        else
            projection = backtrack(projection);
            if projection.solution.isEmpty() then
                return step(projection);
    until undefined == infrastructureNode;
    return projection;
```

Enfin, l'Algorithme 10 essaye de trouver un nœud compatible dans le graphe de l'infrastructure qui respecte la relation d'ordre avec les propriétés du dernier nœud non assigné du graphe d'application. S'il n'en existe pas, l'arbre des solutions atteint une impasse et l'algorithme backtrack sur la pile des solutions et invalide la dernière assignation. Sinon, le nœud trouvé est associé au nœud applicatif qui est alors considéré comme exploré.

Algorithm 11: backtrack

```
input  : projection
output: An updated version of projection
begin
    element = projection.solution.pop() ; projection.exploration.pop() ; if
      undefined != element.parentNode then
        addNodeToExplore(projection, element.parentNode);
    return invalidateLastSolution(projection);
```

L'Algorithme 11 backtrack sur la pile des solutions.

Algorithm 12: invalidateLastSolution

```
input  : projection
output: An updated version of projection
begin
    if !projection.solution.isEmpty() then
        element = projection.solution.last();
        element.infrastructureNode = undefined;
        if projection.exploration.last().applicationNode ==
          element.applicationNode then
            projection.exploration.pop();
        addNodeToExplore(projection, element.applicationNode);
    return projection;
```

L'algorithme 12 invalide le dernier élément de la pile des solutions.

5.1.1.4 Trouver l'ensemble des projections

Il est possible de trouver l'ensemble des projections possibles en appelant de nouveau l'Algorithme 2 jusqu'à ce que la pile des solutions soit vide. L'Algorithme 13 permet de générer cet ensemble de projection.

Algorithm 13: getAllProjections

```
input  : applicationGraph, infrastructureGraph, getFirstNodeIt,
         getNeighboursIt, getCompatibleInfrastructureNodesIt
output: A list of projections
begin
    projectionList = new Array();
    projection = initProjection(applicationGraph, infrastructureGraph,
     getFirstNodeIt, getNeighboursIt, getCompatibleInfrastructureNodesIt)
     repeat
        projection = findNextProjection(projection) ;
         projectionList.push(projection);
    until projection.isEmpty();
    projectionList.pop();
    return projectionList ;
```

5.1.2 Les itérateurs

La sous-section précédente présente une version générique et itérative de l'algorithme de graph-matching. Afin de l'appliquer à notre morphisme de graphe enrichi de l'axiome sur les propriétés des nœuds, nous détaillons dans cette sous section le fonctionnement des trois itérateurs introduits précédemment.

5.1.2.1 Sélection du premier nœud à explorer

La fonction *getFirstNodeIt* prend en paramètre le graphe d'application et retourne un itérateur sur l'ordre de sélection du premier nœud à explorer, qui servira de point de départ à l'exploration de ce graphe. Pour trouver l'ensemble des projections, il suffit de sélectionner un nœud de départ. L'itérateur retournera donc un seul et unique nœud qui peut être choisi aléatoirement, ou selon une heuristique qui dépendra du domaine d'application. Cette heuristique peut par exemple choisir le nœud entité ayant le maximum de propriétés et pour lequel il y aura peu de nœuds correspondants dans l'infrastructure. Ou bien le premier nœud choisi peut être un nœud critique de l'application qui devra avoir été sélectionné au préalable par l'utilisateur (comme une caméra par exemple).

Algorithm 14: getFirstNodeIt

```
input  : applicationGraph
output: An iterator on the first nodes to explore
begin
    nodes = [applicationGraph.getNodes()[0]];
    return new Iterator(nodes);
```

L'Algorithme 14 revoit un itérateur composé d'un seul nœud du graphe d'application, choisi arbitrairement.

5.1.2.2 Sélection des voisins à explorer

L'Algorithme 15 prend en paramètre le graphe d'application, ainsi que le nœud pour lequel l'on souhaite obtenir les voisins et renvoie un itérateur sur les voisins du nœud en question.

Algorithm 15: getNeighboursIt

```
input  : applicationGraph, applicationNode
output: An iterator on the neighbours of the applicationNode
begin
    nodes = applicationGraph.getNeighbours(applicationNode);
    return new Iterator(nodes);
```

5.1.2.3 Sélection des nœuds infrastructures compatibles

L'Algorithme 16 prend en paramètre le nœud applicatif à matcher, le nœud parent, ainsi que la structure de données *projection* et retourne un itérateur sur les nœuds de l'infrastructure qui sont compatibles. Cet itérateur recherche parmi les voisins de l'image du dernier nœud applicatif s'il existe un nœud de l'infrastructure qui respecte la relation d'ordre entre la combinaison des vecteurs de propriétés de tous les nœuds applicatifs qui sont projetés sur celui-ci et son propre vecteur de propriétés. Si cette

relation d'ordre est respectée, le nœud infrastructure peut potentiellement être associé au nœud applicatif. Les fonctions *combine* et *compare* représentent respectivement la loi de composition interne ($\vee_i$) ainsi que la relation d'ordre partielle ($\leqslant_i$).

Algorithm 16: getCompatibleInfrastructureNodesIt

```
input  : projection, parentApplicationNode, applicationNode
output: An iterator on the compatible infrastructure nodes
begin
    compatibleInfrastructureNodes = [];
    applicationProperties = applicationNode.getProperties();
    if undefined == parentApplicationNode then
        infrastructureNodes = projection.infrastructureGraph.getNodes();
    else
        image = getImage(projection, parentApplicationNode);
        infrastructureNodes =
         projection.infrastructureGraph.getNeighbours(image);
    foreach infrastructureNode of infrastructureNodes do
        infrastructureProperties = infrastructureNode.getProperties();
        fibers = getFibers(projection, infrastructureNode);
        foreach applicationNode of fibers do
            applicationProperties = combine(applicationProperties,
             applicationNode.getProperties());
        if compare(applicationProperties, infrastructureProperties then
            compatibleInfrastructureNodes.push(infrastructureNode);
    return new Iterator(compatibleInfrastructureNodes);
```

5.1.3 Déploiement de plusieurs applications

L'algorithme présenté permet de déployer plusieurs applications en même temps. En effet, il est possible de considérer N graphes d'application comme un seul graphe à N composantes connexes. Néanmoins, il est aussi possible de déployer des applications une par une itérativement, en enregistrant le résultat des combinaisons des vecteurs de propriétés et en mettant à jour les propriétés des nœuds du graphe d'infrastructure en conséquence. Comme nous l'avons vu à la section 4.1.1.3, le déploiement d'une nouvelle application ne fait qu'augmenter les contraintes du système. La suppression d'une des applications relâche donc certaines contraintes et ne remet pas en cause le fonctionnement des autres applications qui gardent la qualité de service qu'ils avaient à leur déploiement. La complexité de ce retrait d'application est, dans le pire cas, linéaire avec le nombre de nœuds du graphe d'application associé. Néanmoins, après ce retrait, il est possible que de meilleures solutions, avec de meilleures qualités de services, apparaissent. Si l'objectif est d'adopter des solutions optimum, il faudra donc recalculer de nouveau le déploiement de toutes les applications restantes. Heureusement, bien souvent la solution optimale n'est pas désirée. Seules des solutions respectant une qualité de service minimale sont acceptées.

5.1.4 Exemple illustratif

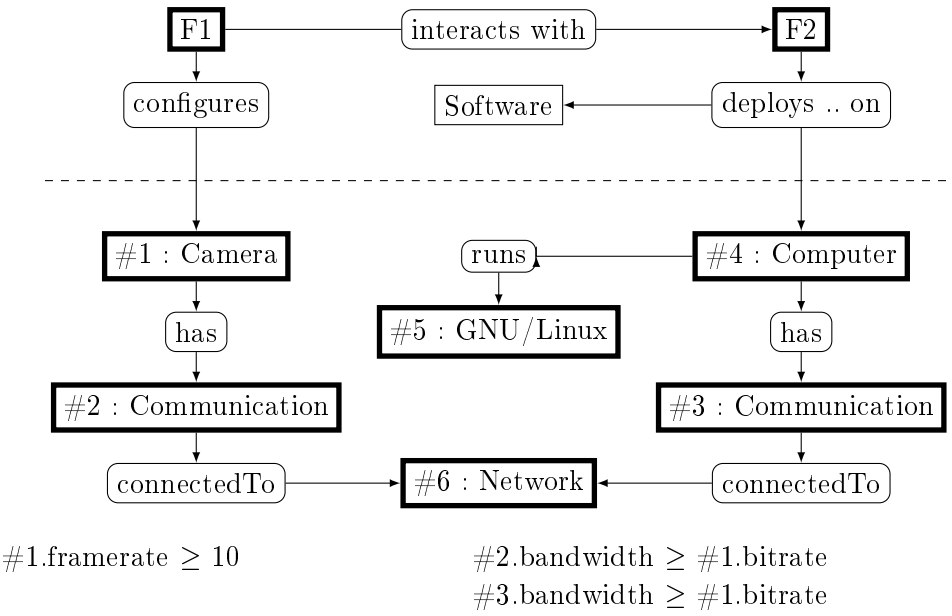


FIGURE 5.1 – Graphe d'application

Cet exemple met en œuvre une application composée de deux fonctionnalités permettant la capture d'un flux d'images provenant d'une caméra, ainsi que son acheminement et son traitement. La Figure 5.1 représente le graphe de cette application. Pour plus de lisibilité, les actions à effectuer pour déployer les fonctionnalités sont représentées par de simples relations. Les propriétés des nœuds sont indiquées sous le graphe. La caméra doit avoir un framerate minimal de 10 images par seconde. L'ordinateur doit exécuter un système d'exploitation de type GNU/Linux. La bande passante est fonction des paramètres de la caméra qui sera choisie. Le graphe de l'infrastructure matérielle qui va accueillir cette application est représenté par la Figure 5.2. Celui-ci est composé de 2 caméras ainsi que de 3 ordinateurs communiquant au travers de deux types de réseaux différents (IP et Bluetooth). Ces entités possèdent un certain nombre de paramètres. Les caméras notamment possèdent un framerate qui est configurable (à choisir parmi un ensemble de framerates possibles). Pour déployer l'application sur cette infrastructure, on applique l'algorithme présenté dans la partie précédente.

La Figure 5.3 représente l'arbre d'exploration des solutions pour le déploiement de l'application sur l'infrastructure. La colonne de gauche représente les nœuds du graphe d'application parcourus en profondeur. La colonne de droite représente les nœuds du graphe d'infrastructure qui ont été choisis. Un nœud barré signifie que les propriétés de celui-ci ne sont pas compatibles avec le nœud applicatif correspondant. Un nœud en rouge correspond à un nœud valide à un moment de l'exécution de l'algorithme, mais qui a été invalidé par la suite. Pour plus de lisibilité, les relations n'ont pas été représentées dans cette figure.

L'algorithme choisit un nœud de départ dans le graphe d'application et essaye de trouver un nœud compatible dans le graphe d'infrastructure. Ici, nous avons choisi la caméra #1 du graphe d'application comme point de départ. Les caméras #1 et#3 du

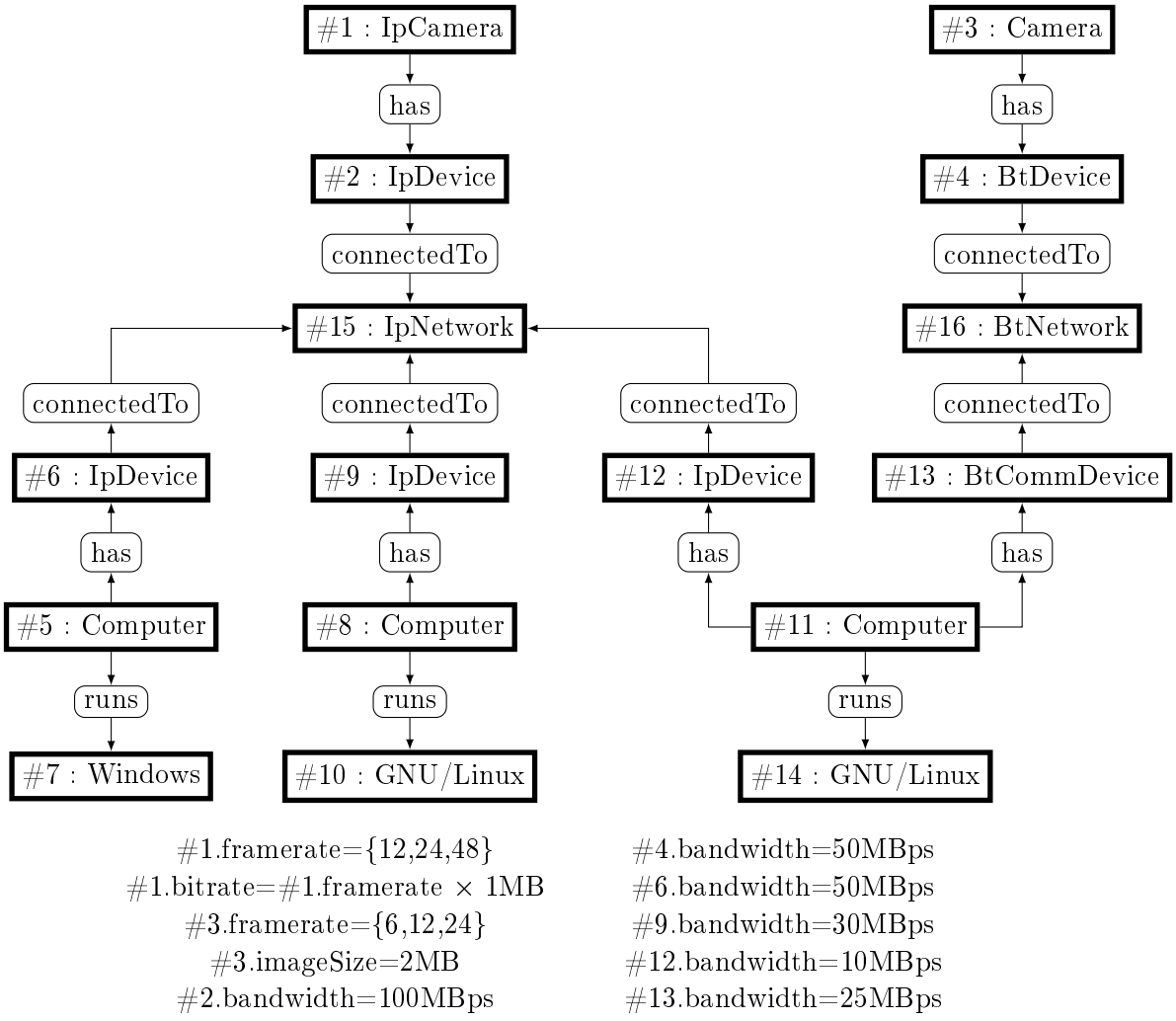


FIGURE 5.2 – Graphe d'infrastructure

graphe d'infrastructure sont compatibles avec ce nœud car elles respectent la relation d'ordre partielle entre leurs propriétés. C'est la caméra IP #1 qui a été choisie. Nous voyons en particulier que l'ordinateur #5 de l'infrastructure matérielle ne peut pas être utilisé à cause de son système d'exploitation (Windows) qui ne respecte pas la relation d'ordre partielle avec la propriété correspondante du nœud du graphe d'application (GNU/Linux). Arrivé à ce point, l'algorithme backtrack et invalide ses assignations précédentes une à une jusqu'à ce qu'il retourne à son dernier point de choix : le nœud #6 *Network* du graphe d'application. L'algorithme choisit d'assigner à ce nœud une nouvelle entité compatible et explore ainsi la nouvelle branche. Ce processus est répété jusqu'à ce qu'un morphisme complet soit trouvé.

Il est intéressant de noter que l'arbre d'exploration de la Figure 5.3 répertorie toutes les solutions possibles et ne s'est pas arrêté à la première trouvée. De plus, dans cet exemple, l'algorithme configure automatiquement le framerate de la caméra pour chacune des solutions trouvées. Dans la première solution, le framerate de la caméra #1 est limité à 12 ou 24 images par seconde à cause de la propriété de bande passante

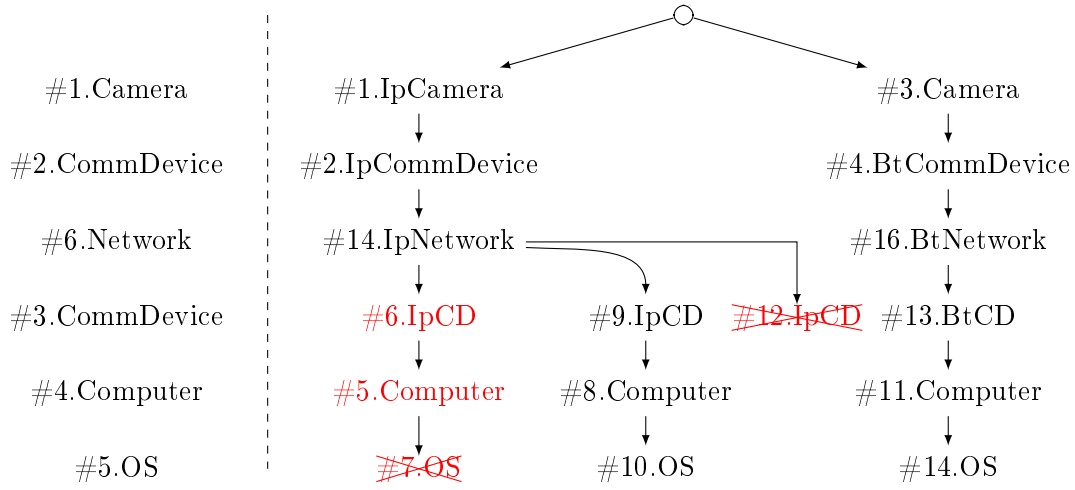


FIGURE 5.3 – Arbre des solutions

entre cette caméra et l'ordinateur hébergeant l'entité logicielle de traitement d'images. De même, ce framerate est limité à 12 images par seconde dans la seconde solution pour les mêmes raisons.

5.2 Génération des projections partielles

Nous venons d'adapter, dans la Section 5.1, un algorithme de Branch and Bound à notre problème d'homomorphisme de graphes enrichi d'un axiome sur les propriétés des nœuds. Néanmoins, cet algorithme est une version centralisée et ne prend pas en compte certaines propriétés importantes des environnements intelligents, comme la confidentialité des informations ou la mise à l'échelle. En effet, l'entité centrale en charge du matching a accès aux graphes d'infrastructure et d'application dans leur totalité et donc aux informations qu'ils contiennent. Pour aller vers une version distribuée de cet algorithme, il nous faut être capable de générer des projections partielles et de compléter ces solutions au fur et à mesure, de manière itérative.

5.2.1 Découpage du graphe d'application

Il est possible de considérer chaque état de l'arbre des solutions retournées par l'Algorithme 6 (*next*) comme une projection partielle potentielle. Cependant, cela n'a pas beaucoup de sens d'utiliser n'importe quelle projection partielle. Comme nous l'avons introduit au Chapitre 3.3.2 présentant le cadre ontologique pour les maisons intelligentes, un périphérique matériel comme un ordinateur ou une caméra, est un regroupement de concepts formant un sous-graphe insécable. Une projection partielle doit donc proposer une projection d'un sous-graphe ou d'un ensemble de ces sous-graphes. Ces périphériques, quelle que soit leur position géographique, communiquent entre eux par des liens de communication au travers de réseaux, au niveau des nœuds *Network*. C'est donc au niveau de ces nœuds communs qu'il est pertinent de couper le graphe des besoins d'une application. La manière de découper une application est

donc dépendante du domaine d'application. Pour générer cette décomposition en sous-graphes, il est possible de réutiliser un algorithme classique de parcours de graphe en n'explorant pas les nœuds voisins d'un nœud commun. Ces sous-graphes obtenus qui représentent les nœuds du nouveau graphe sont ensuite liés les uns aux autres grâce à leurs nœuds $Network$ communs, générant ainsi le graphe des sous-graphes. Deux sous-graphes liés entre eux sont dit adjacents.

La Figure 5.4 représente un patron d'une application. Le résultat de sa décomposition en sous-graphes est illustré à la Figure 5.5. Afin de permettre la représentation d'exemples complexes tout en préservant la lisibilité, les types de nœuds et leurs propriétés associées ont été masqués. Les carrés représentent des nœuds concepts alors que les cercles représentent des nœuds relations. Les nœuds de couleur quant à eux, représentent les nœuds du graphe qui peuvent servir à le découper (par exemple, les nœuds $Network$).

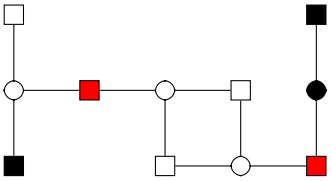


FIGURE 5.4 – Graphe des besoins d'une application

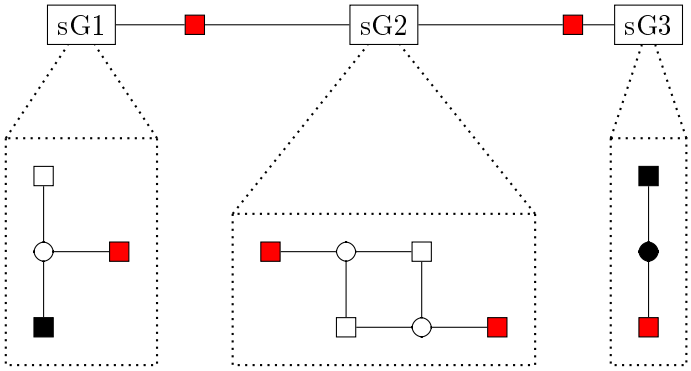


FIGURE 5.5 – Graphe des sous-graphes des besoins d'une application

5.2.2 Projections partielles

Chacun des sous-graphes du patron d'une application représente une partie insécable qui devra être projetée telle quelle sur le graphe d'infrastructure. Nous pouvons maintenant définir une projection partielle comme étant la projection d'un sous-patron ou d'un ensemble connexe de sous-patrons adjacents de l'application sur l'infrastructure matérielle. Pour générer toutes les projections partielles possibles ou simplement trouver les projections complètes, il faut parcourir le graphe des sous-patrons d'une application et appliquer, pour chaque sous-patron, l'algorithme de graph-matching présenté dans la section précédente, en backtrackant si nécessaire. L'itérateur $getFirstNodeIt$ proposera maintenant comme premier nœud à explorer, les nœuds communs qui relient les sous-patrons plutôt qu'un nœud aléatoirement choisi.

5.3 Distribution de l'infrastructure matérielle

Nous cherchons dans cette section à distribuer la représentation du graphe d'infrastructure entre plusieurs entités indépendantes et de faire coopérer ces entités afin de raisonner sur le déploiement d'applications. Cela permet d'isoler des parties de l'infrastructure globale ; chaque entité ne raisonne que sur le sous-graphe qu'il a à sa charge. Nous introduisons ainsi une version distribuée de l'algorithme de graph-matching.

5.3.1 Découpage du graphe d'infrastructure

De la même manière que pour le patron d'une application, le graphe de l'infrastructure peut être découpé au niveau des nœuds réseaux. Nous verrons dans le chapitre suivant que chacun des sous-graphes d'infrastructure pourra être pris en charge par une entité indépendante appelée Agent. Cette distribution du graphe de l'infrastructure permettra de définir des mécanismes pour garantir la confidentialité des ressources présentes dans le système.

La Figure 5.6 représente un exemple de graphe d'infrastructure global. Les cercles et les carrés blancs et noirs représentent des nœuds de types différents. Les nœuds de couleur quant à eux, représentent les nœuds du graphe qui peuvent servir à le découper (par exemple, les nœuds $Network$).

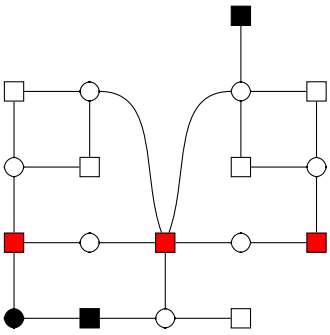


FIGURE 5.6 – Graphe d'infrastructure

La décomposition de ce graphe en sous-graphes est illustrée par la Figure 5.7. Il est intéressant de noter que le graphe d'infrastructure n'est pas obligatoirement découpé au niveau de chaque nœud $Network$. Seul un sous-ensemble de ces nœuds peut servir au découpage. Ainsi, le sous-graphe présenté dans la partie inférieure (sG3) pourrait être encore découpé en trois sous-graphes.

Enfin, à partir de cette décomposition en sous-graphes, il est possible de générer le graphe de ces sous-graphes. Ce graphe, représenté par la Figure 5.8, est bipartite et composé des sous-graphes et des nœuds $Network$ ayant servi à la découpe du graphe global.

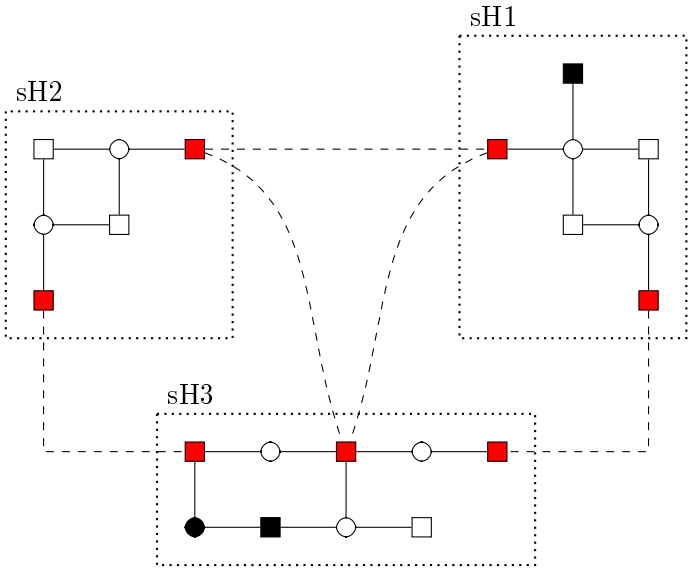


FIGURE 5.7 – Decomposition en sous-graphes

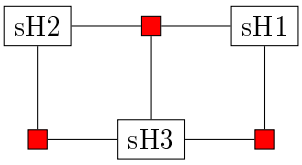


FIGURE 5.8 – Graphe des sous-graphes

5.3.2 Adaptation de l'algorithme

Nous avons vu à la Section 5.2 que le graphe d'application pouvait être décomposé en sous-patrons liés entre eux par des liens d'adjacence (les nœuds $Networks$). À la Sous-section 5.3.1, sur le même principe, nous avons distribué le graphe d'infrastructure en sous-graphes. Pour trouver les projections partielles, il faut donc appliquer l'algorithme de graph-matching présenté à la section 5.1 aux deux super graphes. Pour adapter cet algorithme, il suffit d'adapter le comportement des itérateurs obtenus grâce aux fonctions $getFirstNodeId$, $getNeighboursIt$ et $getCompatibleInfrastructureNodesIt$. Les deux premières fonctions retournent des itérateurs choisissant respectivement le premier nœud du super graphe de l'application à explorer et les voisins d'un nœud application. Ces deux fonctions ne changent pas par rapport à leur définition (Section 5.1.2). Par contre, la dernière fonction, plutôt que de vérifier la relation d'ordre entre les propriétés d'un nœud infrastructure et la combinaison des propriétés des nœuds applications projetés sur ce nœud pour obtenir les nœuds infrastructures compatibles, va appliquer la première version de l'algorithme de graph-matching entre le sous-graphe d'application et un sous-graphe de l'infrastructure. Dans l'Algorithme 17, un nœud compatible représente maintenant un sous-graphe de l'infrastructure pour lequel il existe une solution locale au problème de projection d'un sous-graphe de l'application. Cette solution

représente une projection partielle.

Algorithm 17: getCompatibleInfrastructureNodesIt2

```
input  : projection, parentApplicationNode, applicationNode
output: An iterator on the compatible infrastructure nodes
begin
    projections = [];
    if undefined == parentApplicationNode then
        infrastructureNodes = projection.infrastructureGraph.getNodes();
    else
        image = getImage(projection, parentApplicationNode);
        infrastructureNodes =
          projection.infrastructureGraph.getNeighbours(image);
    foreach infrastructureNode of infrastructureNodes do
        projections.concat(getAllProjection(applicationNode, infrastructureNode,
          it1, it2, it3));
    return new Iterator(projections);
```

Ainsi, l'algorithme de graph matching présenté dans la Section 5.1 intervient maintenant à deux niveaux. Au premier niveau, l'algorithme est appliqué aux super-graphes d'infrastructure et d'application. Chaque super-nœud de l'application est matché à une partie d'un super-nœud de l'infrastructure. Au second niveau, l'algorithme est appliqué entre le super-nœud de l'application et le super-nœud de l'infrastructure pour récupérer cette fois une projection de la partie d'application sur la partie de l'infrastructure.

Nous verrons dans le Chapitre 6 que chacun de ces sous-graphes pourra être pris en charge par une entité indépendante, appelée agent, qui sera chargée de trouver une solution partielle au déploiement d'une application en fonction de son sous-graphe. La résolution de l'algorithme ne faisant que transiter les éléments du patron de l'application qui ont déjà été projetés, la confidentialité du graphe d'infrastructure est donc préservée.

5.3.3 Exemple illustratif

Dans cet exemple, nous allons appliquer l'algorithme de parcours en profondeur des sous-graphes d'infrastructure afin de générer les projections d'une application sur une infrastructure distribuée. Nous reprenons pour ce faire l'exemple présenté dans la section 5.1.4. L'application (Figure 5.1) a été découpée en deux sous-patrons (NA1 et NA2), selon le principe exposé dans la Section 5.2. Le super-graphe de l'application est représenté par la Figure 5.9 dans laquelle les deux sous patrons NA1 et NA2 sont reliés par leur nœud réseau commun ($\#6Network$). Pour plus de lisibilité, les nœuds relations n'ont pas été représentés ici. Ces deux sous-patrons seront à matcher sur le graphe d'infrastructure. Chaque nouvelle association correspondra à une projection partielle. L'infrastructure matérielle (Figure 5.2) quant à elle, a été découpée en trois parties distinctes. La première regroupe les deux caméras ; la seconde, deux des trois ordinateurs ; et la dernière, le dernier ordinateur. Le graphe des sous-graphes de l'in-

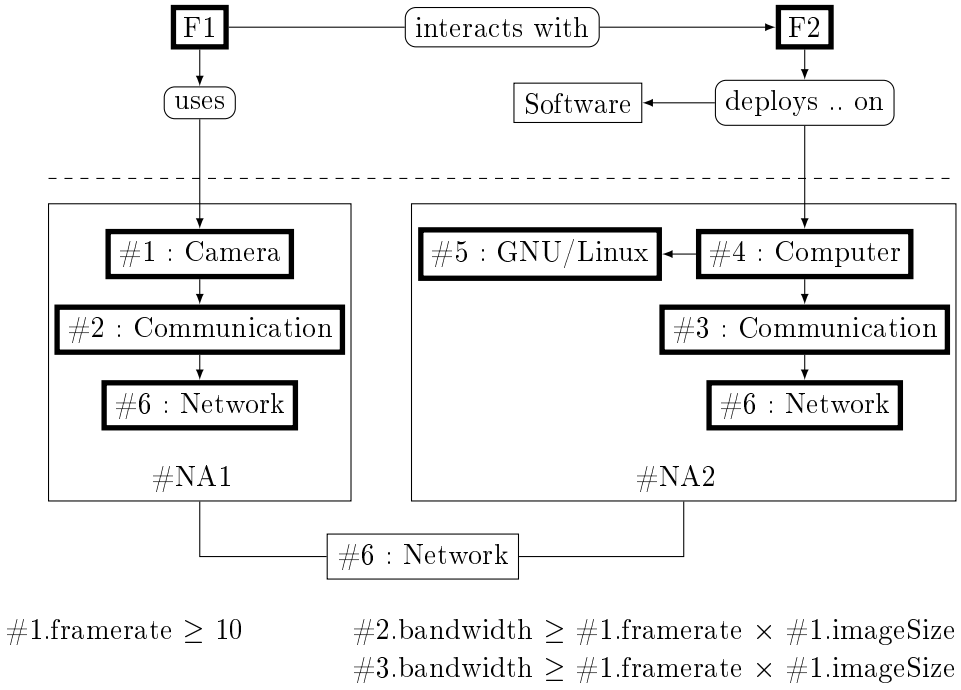
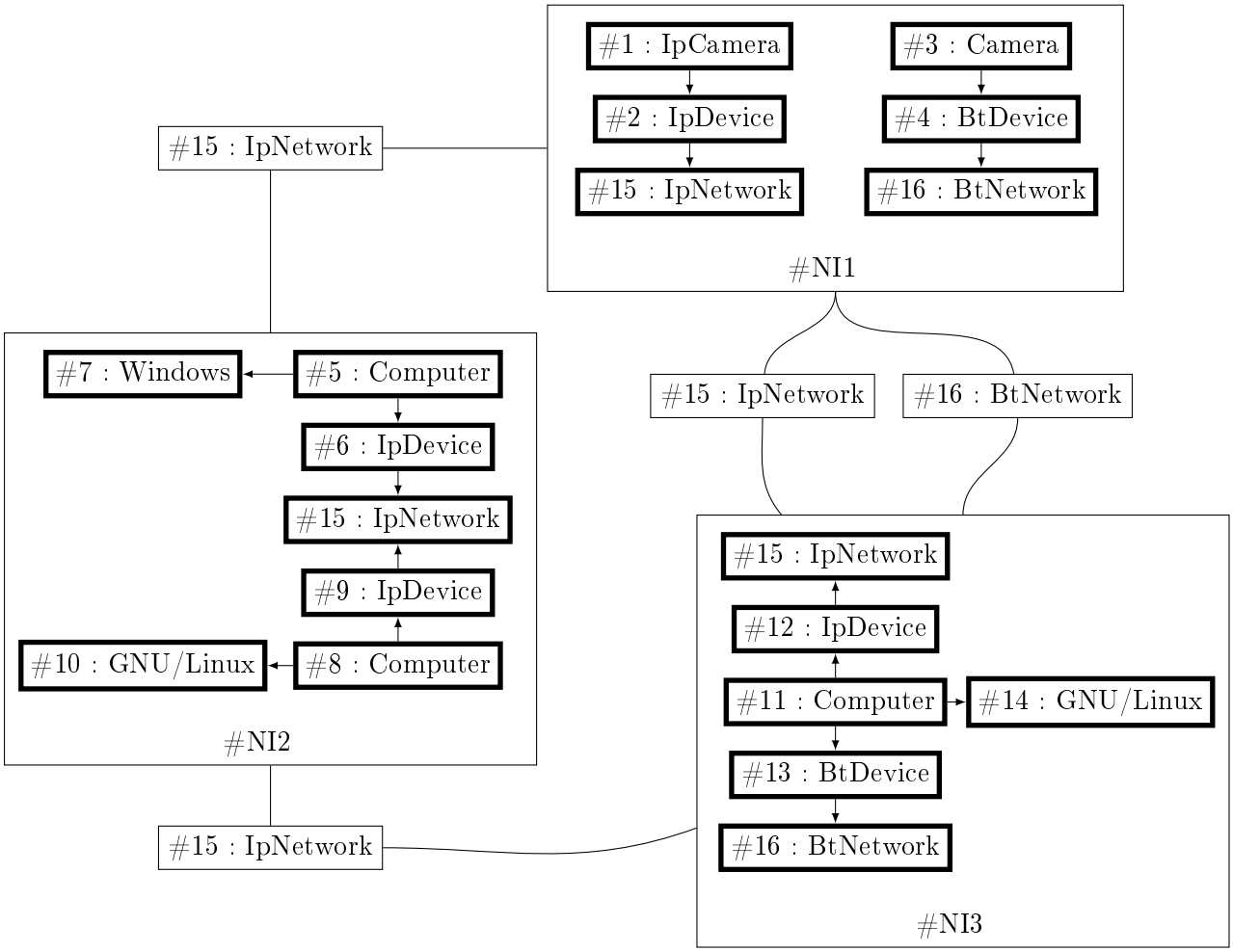


FIGURE 5.9 – Graphe d'application

frastructure est représenté à la Figure 5.10. Ici aussi, chaque sous-graphe est relié à ses sous-graphes adjacents à l'aide des nœuds réseaux communs.

La Figure 5.11 représente l'arbre des solutions. La colonne de gauche représente les deux sous-patrons du graphe d'application avec le détail des besoins matériels de ces nœuds. La colonne de droite représente l'arbre de parcours des sous-graphes d'infrastructure avec le détail des projections partielles. L'algorithme choisit un sous-patron de départ dans le graphe d'application et essaye de trouver des projections partielles dans chacun des sous-graphes de l'infrastructure en appliquant l'algorithme Branch and Bound centralisé (Section 5.1) entre ces deux sous-graphes. Si une projection partielle est trouvée, l'algorithme explore en profondeur le graphe d'application, sélectionne le prochain sous-patron à explorer et essaye de trouver une projection partielle dans un des sous-graphes adjacents, en fixant dans la nouvelle projection partielle, les nœuds *Network* déjà projetés. Si la solution partielle ne peut plus être complétée, alors l'algorithme backtrack jusqu'à trouver une autre projection partielle pour un des sous-patrons de l'application et ainsi explorer une nouvelle branche de l'arbre des solutions. Cet algorithme correspond à l'algorithme Branch and Bound présenté à la Section 5.1, mais appliqué cette fois-ci aux super-graphes.

Dans l'exemple, l'algorithme continue de chercher des solutions même après en avoir trouvé une. L'arbre des solutions nous montre donc l'ensemble des projections de l'application sur l'infrastructure. Comme précédemment, nous obtenons les deux solutions attendues. Dans un vrai système, il n'est pas intéressant de trouver toutes les solutions. Seule la première projection importe.



#1.framerate={12,24,48}
#1.bitrate=#1.framerate × 1MB
#3.framerate={6,12,24}
#3.bitrate=#1.framerate × 2MB
#2.bandwidth=100MBps
#4.bandwidth=50MBps
#6.bandwidth=50MBps
#9.bandwidth=30MBps
#12.bandwidth=10MBps
#13.bandwidth=25MBps

FIGURE 5.10 – Graphe d'infrastructure

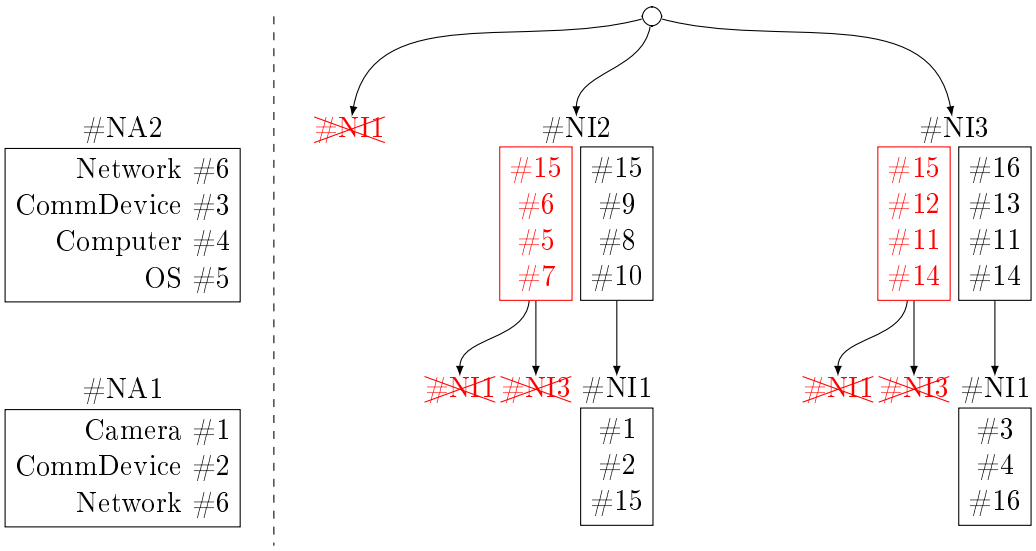


FIGURE 5.11 – Arbre des solutions

5.4 Conclusion

Dans ce chapitre, j'ai proposé d'utiliser un algorithme de Graph-Matching pour de trouver les projections des besoins des applications sur l'infrastructure matérielle afin de résoudre le problème du déploiement d'application. L'algorithme retenu est une adaptation d'un algorithme de type Branch and Bound appliqué à notre problème. Après avoir expliqué la manière dont il était souhaitable de distribuer le graphe d'infrastructure, j'ai montré qu'il était possible de résoudre le problème du déploiement de manière distribuée, en utilisant l'algorithme à deux niveaux : au niveau des sous-graphes d'infrastructure et d'applications et au niveau des super-graphes eux-mêmes. Cet algorithme permet de préserver la confidentialité du graphe de l'infrastructure en ne communiquant que les nœuds du patron de l'application qui ont été trouvés. Cet algorithme distribué sera utilisé au chapitre suivant par des agents qui coopéreront pour trouver une projection des besoins d'une application sur l'infrastructure, avant de déployer effectivement cette application.

Chapitre 6

Modélisation multi-agents

Sommaire

Dans les chapitres précédents, nous avons présenté un moyen de modéliser, à l'aide de graphes, l'infrastructure matérielle disponible et les applications déployables dans des systèmes ambiants. Nous avons proposé de raisonner sur ces graphes afin de trouver dans l'infrastructure, les entités matérielles qui supportent l'exécution des applications en utilisant un algorithme de matching de graphes dans une version distribuée. Ceci nous permet de bénéficier des avantages apportés par l'hétérogénéité matérielle afin de déployer des applications. Néanmoins, afin de gérer l'infrastructure d'une ville entière, il nous faut bien plus qu'une représentation des connaissances et un algorithme. Il nous faut, en plus de ces mécanismes de raisonnement et de déploiement, prendre en compte d'autres points, comme la dynamicité des environnements, la gestion du contexte, la gestion de la confidentialité des ressources, le passage à l'échelle, la robustesse du système, les interactions avec les utilisateurs, ou le déploiement en lui-même, etc.

La dynamicité des environnements implique qu'il faille pouvoir détecter les modifications de l'environnement, vérifier que les applications déployées soient toujours consistantes et planifier un redéploiement dans le cas contraire. Ces modifications peuvent être dûes à un ajout ou un retrait d'entité matérielle dans l'infrastructure, la mise à jour d'une propriété d'un des nœuds du graphe, ou la prise en compte d'informations contextuelles (localisation de l'utilisateur, bande passante disponible, etc.). Des mécanismes permettant de renforcer la confidentialité des ressources des utilisateurs doivent pouvoir être proposés. Il n'est pas pensable que quiconque ait accès aux ressources présentes au sein d'un logement, sauf si leur propriétaire l'ait autorisé. La composition de l'infrastructure matérielle ne devrait même pas être accessible de l'extérieur. Le passage à l'échelle ainsi que la robustesse du système sont des propriétés essentielles pour passer de la gestion d'un logement à la gestion d'une ville entière. Enfin, les interactions avec les utilisateurs ainsi qu'avec l'environnement réel doivent être prises en charge afin de pouvoir mettre à jour en temps réel les informations sur le système, ainsi que le contexte d'exécution des applications.

Pour répondre à ces problèmes, nous pensons que les systèmes distribués intelligents et en particulier les systèmes multi-agents, conviennent tout à fait à notre approche. En effet, le paradigme agent fournit de bonnes propriétés qui permettent de faciliter le traitement local des informations, de garantir l'autonomie du système et de gérer différentes parties de la confidentialité. Les systèmes multi-agents ont montré leur efficacité pour la conception de systèmes distribués, complexes et robustes. Le paradigme agent permet d'imaginer des systèmes dans lesquels les agents sont autonomes, possèdent des objectifs à atteindre et peuvent interagir aussi bien entre eux qu'avec leur environnement.

Nous allons voir dans ce chapitre comment utiliser et organiser un système multi-agent afin de gérer l'infrastructure matérielle ainsi que le déploiement d'applications dans le but de garantir à l'utilisateur la protection des ressources. La conception d'un tel système doit permettre d'établir les types d'agents, leurs rôles, leurs interactions et leur fonctionnement interne. Cette conception passe par plusieurs étapes : Une première étape d'analyse des besoins permet, à partir de cas d'usages et de scénarios (Chapitre 3.1) d'isoler les différentes fonctions du système ainsi que ses besoins et contraintes (Section 6.1). À partir de cette analyse, il est possible de définir l'architecture du SMA (Section 6.2) en terme de types d'agents et d'artifacts, de rôles, d'organisations et d'autorisations. Une fois les rôles et les types d'agents définis, il faut dresser un modèle

d'interaction permettant d'établir les protocoles d'interactions entre les agents et les artifacts (Section 6.3). Enfin, il est alors possible de détailler le fonctionnement interne de chaque agent sous la forme de buts et de plans (Section 6.4).

6.1 Identification des besoins

Le scénario du portier vidéo présenté au Chapitre 3.1 présente plusieurs cas de déploiement d'une application. Dans le premier cas, la maison, consciente de l'emplacement de l'utilisateur choisit un écran disponible à côté de celui-ci. Dans le second cas, la localisation de l'utilisateur étant inconnue, une notification est envoyée sur un périphérique par défaut, à savoir, son smartphone. Enfin, dans le dernier cas, c'est la télévision de son voisin qui est utilisée, l'utilisateur ayant reçu le droit de l'utiliser même si elle ne lui appartient pas.

À partir de ce scénario, il est possible d'isoler les cinq besoins principaux du système :

1. Le système doit d'abord trouver les entités matérielles à utiliser pour lancer une application. Ces entités doivent respecter des contraintes matérielles, ainsi que des contraintes contextuelles telles que l'emplacement de l'utilisateur.
2. Une fois les entités matérielles sélectionnées, le système doit pouvoir déployer l'application ou des parties de l'application sur l'infrastructure. Dans le scénario, l'application du portier vidéo se divise en deux parties. La première partie s'occupe de monitorer la sonnette de la maison. Elle est déployée automatiquement lorsque l'utilisateur décide d'utiliser l'application du portier vidéo. Lorsqu'un individu actionne la sonnette, la seconde partie de l'application qui consiste à afficher l'image de la caméra sur un écran est déployée. C'est le déploiement de cette seconde partie qui est décrite dans le scénario.
3. Le système doit pouvoir désinstaller une application ou une de ses parties. Dans le scénario, lorsque l'utilisateur coupe la communication avec l'invité, la seconde partie de l'application du portier vidéo est désinstallée et les ressources utilisées sont libérées.
4. Le système doit pouvoir monitorer l'environnement, c'est-à-dire récupérer des informations contextuelles sur l'emplacement d'un utilisateur ou la quantité de bande passante disponible par exemple. Si une inconsistance entre l'état de l'infrastructure et les besoins des applications est détectée, un redéploiement de l'application ou d'une de ses parties doit pouvoir être déclenché.
5. Enfin, un utilisateur doit pouvoir gérer les entités matérielles dont il est propriétaire. Il peut utiliser le matériel d'autres utilisateurs s'il y est autorisé. Mais il ne doit pas avoir de connaissances sur les parties de l'infrastructure matérielle qui ne lui appartiennent pas.

Le premier point a été traité dans les Chapitres 3, 4, 5. Un algorithme distribué a été proposé afin de trouver les entités matérielles de l'infrastructure qui supporterait le déploiement d'une application en se basant sur leur description. Les second et troisième points impliquent qu'il faille interagir avec l'environnement réel afin de configurer les entités matérielles du système. Le quatrième besoin demande aussi une interaction

avec l'environnement afin d'en percevoir ses caractéristiques et les propriétés. Enfin le dernier point soulève des questions en rapport avec la confidentialité des ressources des utilisateurs. Comment garantir que le système appartenant à un utilisateur n'ait pas accès aux informations concernant l'infrastructure d'un autre utilisateur ?

Le système multi-agents que nous souhaitons modéliser est utilisé afin de déployer les applications sur une infrastructure matérielle existante, de contrôler l'état du système ambiant et de maintenir sa cohérence. Les agents du système permettent d'effectuer les raisonnements de haut niveau, alors que des artifacts font office d'outils pour interagir et percevoir l'environnement. L'aspect décentralisé des architectures multi-agents est un atout pour sécuriser les données et les ressources. Tout d'abord, la représentation de l'infrastructure peut être divisée et répartie entre les agents afin qu'aucune connaissance globale de l'infrastructure n'existe. Ceci encourage donc les raisonnements locaux pour le déploiement d'applications. Ensuite, ces agents doivent être organisés, de manière hiérarchique ou holonique, afin de permettre d'établir des politiques de protection des ressources des utilisateurs et de guider le raisonnement local pour le déploiement d'applications. Enfin, la coopération entre ces agents permet de trouver une solution au problème du déploiement, même si aucune solution locale n'existe.

A partir de ces observations, il est possible d'établir les différents rôles du système :

1. Interagir avec l'utilisateur
2. Maintenir la cohérence d'une application
3. Trouver des projections des besoins d'une application sur l'infrastructure
4. Interagir avec l'environnement pour configurer des entités matérielles
5. Percevoir les propriétés de l'environnement
6. Mettre à jour le graphe de l'infrastructure
7. Gérer les niveaux d'autorisation et de partage des ressources

Ces rôles seront attribués aux différentes entités du système dans la section suivante.

6.2 Architecture du SMA

6.2.1 Agentification

Le processus de déploiement implique qu'un utilisateur veuille déployer une application sur une infrastructure matérielle. Afin d'avoir une séparation claire entre ces trois parties, nous proposons donc trois classes d'agents : les agents infrastructure, les agents application et les agents utilisateur. Une quatrième classe d'agents a été rajoutée pour permettre la création d'organisations d'agents : les super agents infrastructure.

Un **agent infrastructure** gère une partie de l'infrastructure matérielle. Il possède la description graphe de cette partie de l'infrastructure : les entités matérielles, leurs propriétés et leurs relations. Cette description de l'infrastructure correspond à la représentation de l'environnement de l'agent. Cette représentation n'est jamais transmise à d'autres agents. Ainsi, pour déployer une application, un agent infrastructure

devra trouver une projection partielle de l'application (rôle 3) et coopérer avec d'autres agents infrastructure pour compléter cette projection. Ce graphe de l'infrastructure est mise à jour (rôle 6) lorsque des informations sur l'état réel de l'infrastructure lui sont transmises. Enfin, comme la description de l'infrastructure n'est pas partagée entre les agents, ce sont les agents infrastructure qui doivent gérer les différents niveaux d'autorisation et de partage des ressources (rôle 7).

Pour sa part, un **agent application** vise à maintenir la cohérence d'une application durant son exécution (rôle 2). Si les besoins matériels de l'application ne sont plus respectés suite à un changement dans l'infrastructure matérielle, alors l'agent application associé replanifie un déploiement de la partie logicielle concernée en interagissant avec les agents infrastructure concernés.

Un **agent utilisateur** est attaché à un utilisateur et mémorise ses préférences, sa position, son contexte. C'est l'interface privilégiée entre l'utilisateur et le système multi-agents (rôle 1). Un utilisateur est donc représenté au sein du système à l'aide de son agent personnel. Celui-ci réceptionne les requêtes de déploiement d'applications de la part de l'utilisateur et crée les agents application associés qui vont interagir avec les agents infrastructure autorisés afin de déployer l'application.

Les agents infrastructure peuvent être groupés, formant ainsi des sous systèmes multi-agents. Ces groupes sont représentés par un **super agent infrastructure**. D'un point de vue extérieur, un super agent infrastructure est vu comme un agent infrastructure normal. Celui-ci agit comme un proxy entre les agents du groupe et ceux à l'extérieur du groupe. Il est alors plus facile d'abstraire des groupes d'agents et de les rendre invisibles de l'extérieur. Il en résulte une organisation multi-échelle qui contribue à la confidentialité des ressources en cachant les informations concernant la structure de ses sous-organisations.

Ces différentes classes d'agents ne font pas nécessairement partie de l'infrastructure matérielle prise en charge. Ells peuvent s'exécuter sur n'importe quelle entité matérielle compatible, que celle-ci soit incluse ou non dans l'infrastructure.

En plus de ces quatre classes d'agents, nous définissons également deux classes d'artifacts, qui sont des outils utilisables par les agents pour interagir physiquement avec l'environnement [Ricci 03, Ricci 11] : les artifacts de déploiement, et les artifacts de monitoring.

Les **artifacts de déploiement** [Flissi 08] peuvent être utilisés par les agents infrastructure afin de déployer ou de désinstaller effectivement des fonctionnalités d'une application sur certaines entités matérielles, ou de configurer ces entités matérielles afin qu'elles soient utilisables par les applications (rôle 4).

Les **artifacts de monitoring** fournissent des informations contextuelles au SMA (rôle 5), comme la position d'un utilisateur ou la quantité de bande passante disponible sur un canal de communication. Ces informations aident les agents à garder leurs graphes de l'infrastructure ou de l'application à jour.

6.2.2 Politiques de partage

Afin d'améliorer la confidentialité des ressources, nous proposons aussi des **politiques de partage**. Un agent utilisateur peut être autorisé par le propriétaire d'une partie de l'infrastructure matérielle à utiliser les entités de cette infrastructure et de coopérer avec les agents infrastructure ou les super agents associés pour déployer des applications. Si un utilisateur est inconnu d'un agent infrastructure, il ne pourra alors pas utiliser les ressources proposées par cet agent. Dans le cas contraire, chaque agent utilisateur peut être autorisé à accéder et à interagir avec ces groupes à l'aide de différents niveaux d'autorisation. Ces niveaux d'autorisation sont définis par les propriétaires de chaque agent ou super agent infrastructure. Seul le niveau administrateur est fixé et permet d'identifier le ou les propriétaires du système. Ce niveau d'autorisation permet à l'utilisateur de configurer les autres niveaux de partage, de reconfigurer ou de créer des sous-organisations à partir de l'agent ou du super agent infrastructure et d'avoir accès à toutes les ressources gérées par cet agent. Lorsqu'un administrateur crée une sous-organisation, il devient automatiquement administrateur des nouveaux agents infrastructure créés.

6.2.3 Déploiement

Le processus de déploiement est initié par l'utilisateur au travers de son agent utilisateur. Celui-ci va créer un agent application en charge du déploiement et du maintien de la cohérence. Cet agent choisit un agent infrastructure ou un super agent infrastructure, en fonction du contexte et lui émet une requête pour déployer l'application. L'agent infrastructure (ou super agent) concerné essaye de résoudre le problème du déploiement à l'aide de l'algorithme de matching de graphe présenté au chapitre précédent et retourne une solution complète ou partielle. Si une solution partielle est retournée, l'agent application demande alors à d'autres agents ou super agent de continuer la résolution du problème. Lorsqu'une projection complète est trouvée et validée par l'agent application, les agents infrastructure ayant participé à l'élaboration de la projection vont donc déployer effectivement les parties de l'application en utilisant les artifacts de déploiement. Les artifacts de monitoring de l'environnement permettent aux agents infrastructure de connaître l'état réel de l'environnement. Lorsque certaines propriétés de l'environnement changent, les agents infrastructure notifient les agents application concernés par ces changements. Ceux-ci peuvent ainsi décider si l'application est toujours fonctionnelle ou si au contraire, il y a une inconsistance et s'il faut planifier un redéploiement de certaines parties de l'application.

6.2.4 Scénario du portier vidéo

La Figure 6.1 représente l'organisation multi-agents appliquée au scénario du portier vidéo. Le graphe de l'infrastructure matérielle a été réparti entre trois agents infrastructure. Le premier prend en charge les entités matérielles de la salle de vie de M. Snow (comme la télévision). Le second s'occupe des entités de la salle de bain (comme le miroir connecté). Ces deux agents forment un groupe grâce à un super agent infrastructure

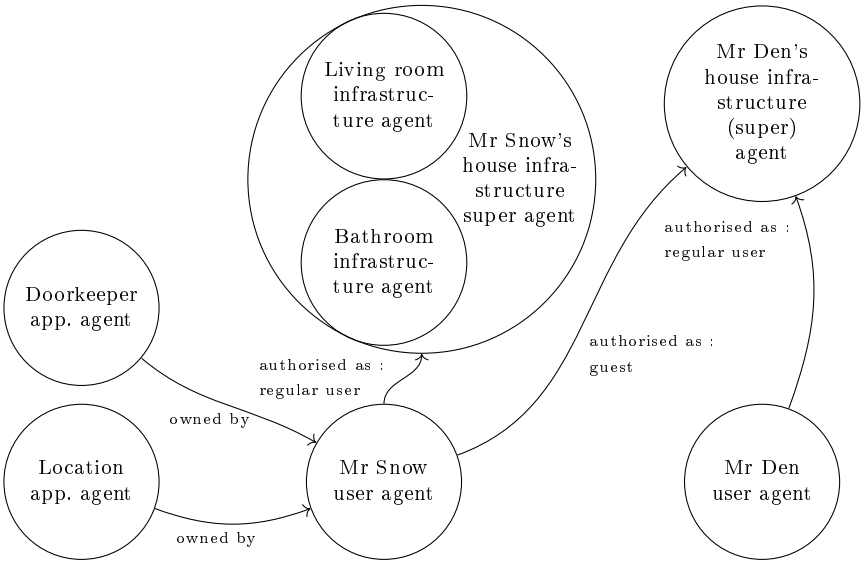


FIGURE 6.1 – Organisation agents : scénario portier vidéo

représentant le logement de M. Snow. Enfin, un troisième agent infrastructure s'occupe de la maison de M. Den, son voisin. Ce troisième agent infrastrucure peut être en réalité un super agent infrastructure regroupant lui-même d'autres agents gérant plus finement le logement de M. Den. Ainsi, M. Snow peut dialoguer avec ce super agent sans connaître le nombre d'agents gérant le logement de M. Den, et réciproquement. Le scénario met en œuvre deux agents application. Le premier prend en charge l'application portier vidéo ; lorsque quelqu'un sonne à la porte du logement de M. Snow, cet agent application déclenche le déploiement de la fonctionnalité d'interaction vidéo. Le second agent application gère l'application de localisation d'un utilisateur chez M. Snow. Cette information contextuelle est remontée aux agents infrastructure de M. Snow et est utile pour le déploiement d'autres applications, comme le portier vidéo. Enfin, ce scénario met en œuvre deux agents utilisateur ; un pour chaque protagoniste. Chacun de ces utilisateurs possède les autorisations nécessaires pour utiliser les ressources de leur logement respectif. Dans ce scénario, trois types d'autorisation sont définis : (1) le niveau administrateur, utilisateur et invité. Lorsqu'un agent utilisateur est administrateur vis-à-vis d'un agent infrastructure, il a accès à toutes les ressources proposées par celui-ci, il peut modifier la politique de partage ainsi que l'organisation de l'infrastructure. Avec un niveau utilisateur, l'agent utilisateur a un accès à l'ensemble des ressources proposées par l'agent infrastructure, mais il ne peut plus gérer cette infrastructure, ni les niveaux de partage. Enfin, avec le niveau invité, l'utilisateur a un accès restreint aux ressources de l'agent infrastructure ; seules celles considérées comme étant non critique sont autorisées à être utilisées. Ainsi, l'agent utilisateur de M. Snow est considéré comme un "utilisateur" auprès du super agent infrastructure gérant le logement de M. Snow. Il en est de même pour M. Den. M Snow est aussi considéré comme "invité" par le super agent infrastructure gérant le logement de M. Den. À ce titre, il est autorisé à utiliser certaines entités considérées comme non critique par M. Den, à savoir, sa télévision notamment.

6.2.5 Synthèse

Dans cette section, nous venons de définir l'architecture du système multi-agent. Des agents utilisateur, application et infrastructure ont été définis afin d'obtenir une séparation claire entre les différentes parties du système. Les super agents infrastructure ont été introduits afin de permettre la création d'organisations hiérarchiques d'agents infrastructure. Des politiques de partage de ressources ont été introduites afin de contrôler l'utilisation et le partage des ressources. L'attribution des autorisations basée sur ces politiques de partage détermine donc l'organisation et les accointances des agents du SMA.

Cette décomposition agent permet d'encapsuler une partie des mécanismes garantissant la confidentialité des ressources. Tout d'abord, les agents infrastructure ne partagent pas les connaissances qu'ils ont de leur infrastructure matérielle. Chacun de ces agents est autonome est possède une vision locale de l'infrastructure. La structure des sous-infrastructures et les propriétés des entités matérielles qu'ils contiennent ne sont donc connues que d'un seul agent infrastructure. Ensuite, la hiérarchie des agents infrastructure, définie par les super agents infrastructure, permet de garder cachés les détails de l'organisation des agents. Enfin, la politique de partage autorise ou interdit l'utilisation de ressources par les agents utilisateur. Toutes ces mesures au niveau architectural permet d'améliorer la confidentialité des ressources.

6.3 Interactions entre agents

Dans les sections précédentes, nous avons décrit les différents types d'agents/artifacts composant le système multi-agents, ainsi que leurs rôles. Cette section a pour but de présenter les interactions entre ces entités, en fonction de leurs rôles.

6.3.1 L'agent utilisateur

Le déploiement d'une application sur l'infrastructure est à l'initiative de l'utilisateur, au travers de son agent utilisateur. La Figure 6.2 représente les interactions entre l'utilisateur, son agent utilisateur et l'agent application pour le lancement d'une application. Lorsque l'utilisateur décide de lancer une nouvelle application, l'agent utilisateur crée l'agent application associé. Une fois créé, cet agent notifiera l'agent utilisateur et entamera la procédure de déploiement.

La Figure 6.3 illustre le cas où l'utilisateur veut désinstaller l'application. Dans ce cas, l'agent utilisateur envoie une requête de désinstallation de toute l'application à l'agent application. Une fois que ce dernier a fini sa tâche, l'agent utilisateur tue l'agent application.

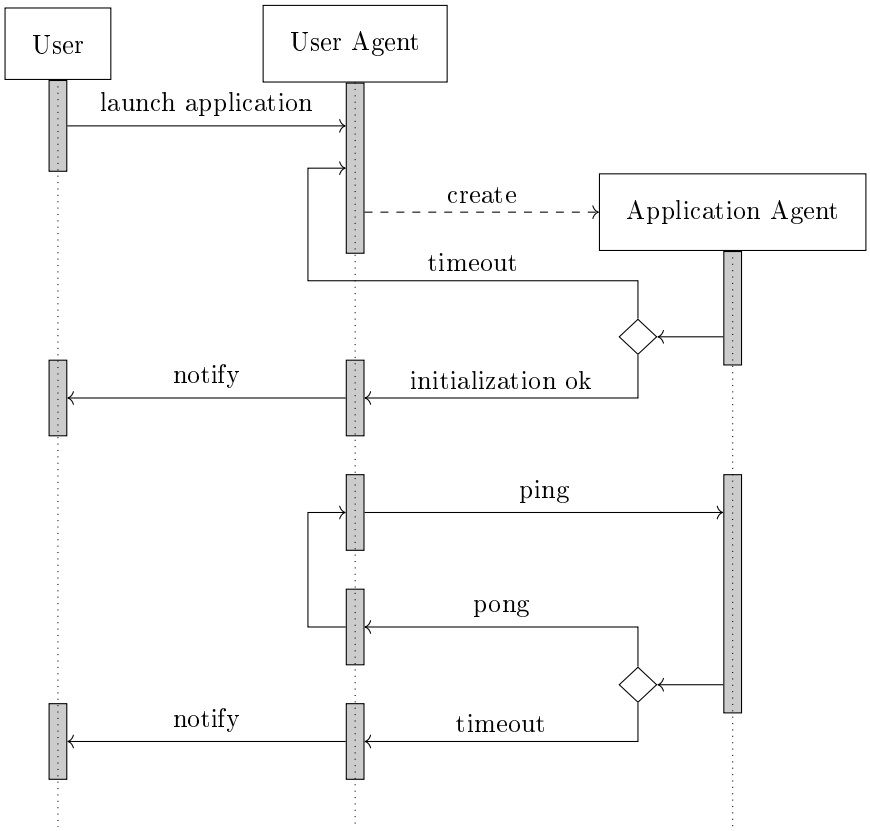


FIGURE 6.2 – Agent Utilisateur : lancement d'une application

6.3.2 L'agent application

L'agent application interagit avec les agents infrastructure afin de déployer ou de désinstaller l'application.

La Figure 6.4 montre les interactions entre l'agent application et les agents infrastructure dans le cas du déploiement. L'agent application interroge successivement tous les agents de son voisinage jusqu'à ce que l'un d'eux trouve une projection valide. Lorsque c'est le cas, l'agent application va demander à cet agent infrastructure de déployer effectivement l'application. Lorsque le déploiement est terminé, l'agent utilisateur en charge de l'application est notifié.

La Figure 6.5 montre les interactions, centrées autour de l'agent application, dans le cas d'une désinstallation. L'agent application va demander aux agents infrastructure en charge du déploiement de, cette fois-ci, prendre en charge la désinstallation de l'application.

6.3.3 L'agent infrastructure

Afin de trouver une projection, de déployer ou de désinstaller une application, les agents infrastructure coopèrent entre eux.

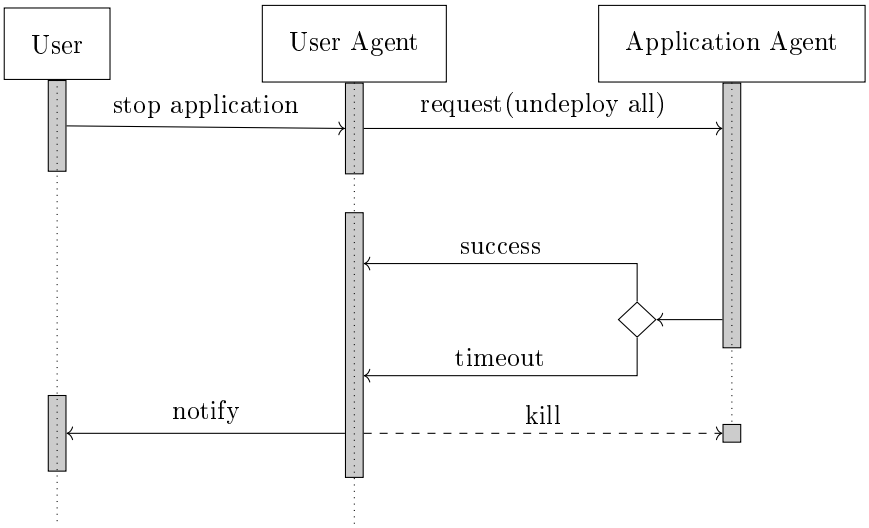


FIGURE 6.3 – Agent Utilisateur : arrêt d'une application

La Figure 6.6 montre les interactions entre les agents infrastructure pour trouver une projection des besoins de l'application sur l'infrastructure. L'agent application va d'abord interroger un agent infrastructure parmi ceux qu'il connaît. Si ce dernier n'est pas capable de trouver une projection partielle, un autre agent est interrogé. Si l'agent infrastructure trouve une solution partielle, il va alors interroger un agent voisin et lui demander de compléter la solution. Si aucune solution n'est trouvée, l'agent infrastructure essayera de trouver une autre solution partielle à tester et recommencera le processus. Enfin, si une solution complète est trouvée, chaque agent infrastructure mémorise la solution et lui attribue un identifiant qui sera communiqué à l'agent application à l'origine de la requête.

Une fois une projection de l'application trouvée, il est possible de demander le déploiement effectif de celle-ci. Dans la Figure 6.7, l'agent application demande à l'agent infrastructure qui a trouvé la projection (l'agent #2), de déployer l'application. Cet agent va donc demander à ses artifacts de déployer certaines parties de l'application. Pour les autres parties, l'agent infrastructure va demander aux autres agents qui ont coopéré à la recherche d'une projection, de continuer le déploiement. Ces autres agents vont à leur tour soit, déléguer le déploiement à d'autres agents infrastructure soit, utiliser des artifacts pour déployer des parties de l'application sur l'infrastructure de l'agent. Le protocole d'interaction est le même dans le cas de la désinstallation d'une partie de l'application.

6.3.4 Exemple illustratif

Nous avions présenté un exemple pour illustrer l'algorithme de graph-matching distribué à la Section 5.3.3. Nous reprenons ici cet exemple afin de présenter les interactions entre les agents application et infrastructure Le graphe de l'infrastructure était décomposé en trois sous-graphes (Figure 5.10). Chacun de ces sous-graphes est pris en charge par un agent infrastructure (numéroté de 1 à 3). L'application, composée de deux sous-graphes (Figure 5.9), est prise en charge par un agent application.

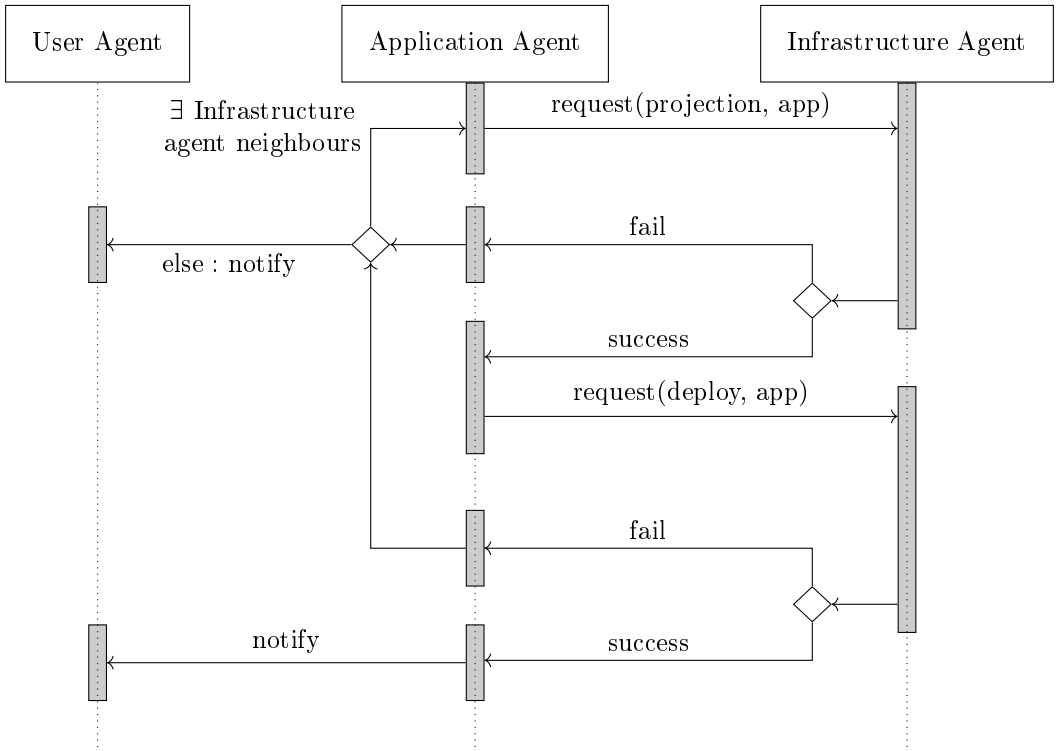


FIGURE 6.4 – Agent Application : Déploiement d'une partie de l'application

La Figure 6.8 représente le diagramme d'interactions, centré sur l'agent application, lors du déploiement de l'application. Les interactions entre les agents infrastructure ne sont pas représentées ici. L'ordre des interactions suit l'arbre des solutions précédemment détaillé (Figure 5.11).

La Figure 6.9 décrit le diagramme d'interactions des agents infrastructure afin de trouver une projection de l'application sur l'infrastructure matérielle. L'agent application va d'abord interroger l'agent infrastructure #1. Ce dernier n'est pas capable de trouver une projection pour le sous-graphe NA2. L'agent application va donc demander à un autre agent infrastructure, le #2. Ici, cet agent trouve une solution partielle de NA2 vers son sous-graphe d'infrastructure. Il va donc demander aux agents de son voisinage de continuer la solution partielle en essayant de projeter le sous-graphe NA1. Les agents infrastructure #1 et #3 sont interrogés successivement et aucun ne trouve de projection. L'agent infrastructure #2 a parcouru tous ses voisins sans trouver de solution. Il va donc chercher une autre solution partielle pour NA2 et recommencer le processus. Il interroge l'agent infrastructure #1 qui trouve une solution pour NA1. La projection étant alors complète, chaque agent infrastructure mémorise la solution et lui attribue un identifiant qui sera communiqué à l'agent application à l'origine de la requête.

Enfin, la Figure 6.10 présente les interactions entre les agents infrastructure et les artifacts de déploiement pour déployer effectivement l'application de manière distribuée.

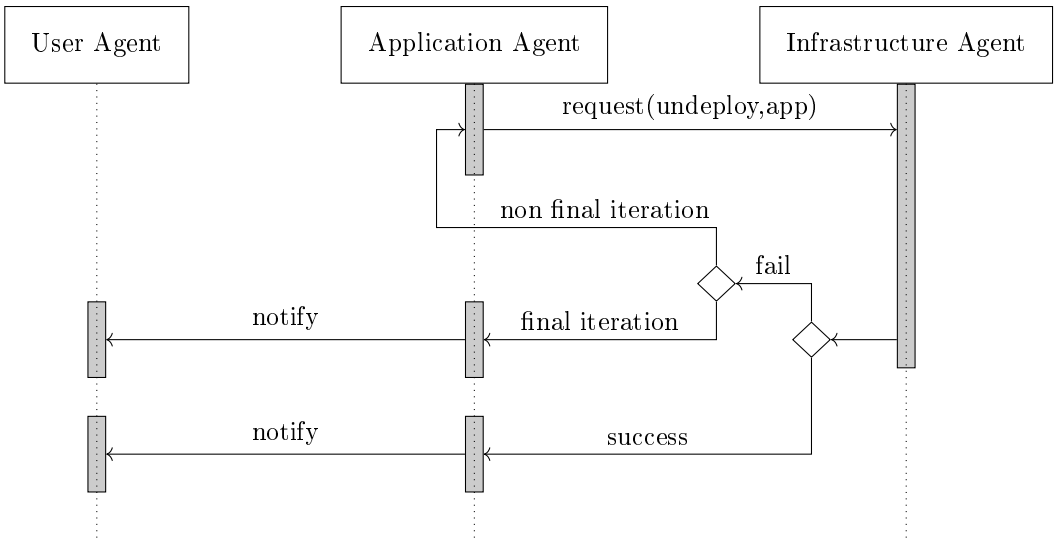


FIGURE 6.5 – Agent Application : Désinstallation d'une partie de l'application

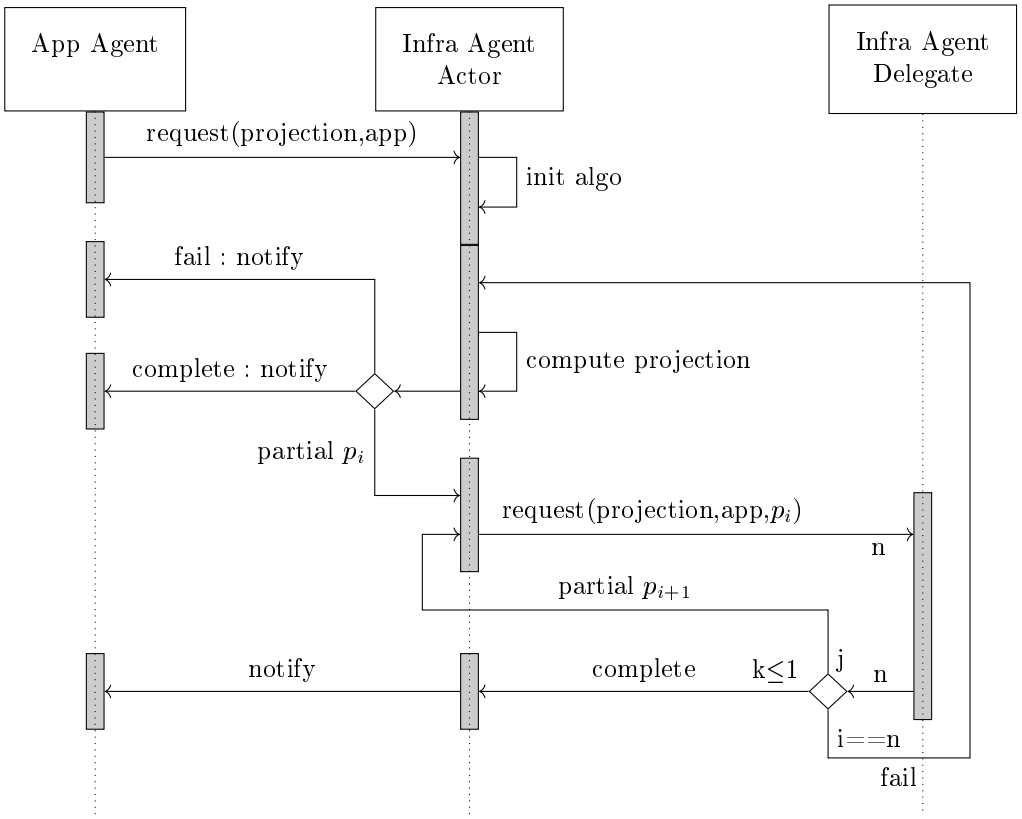


FIGURE 6.6 – Agent Infrastructure : Obtenir une projection d'une partie de l'application

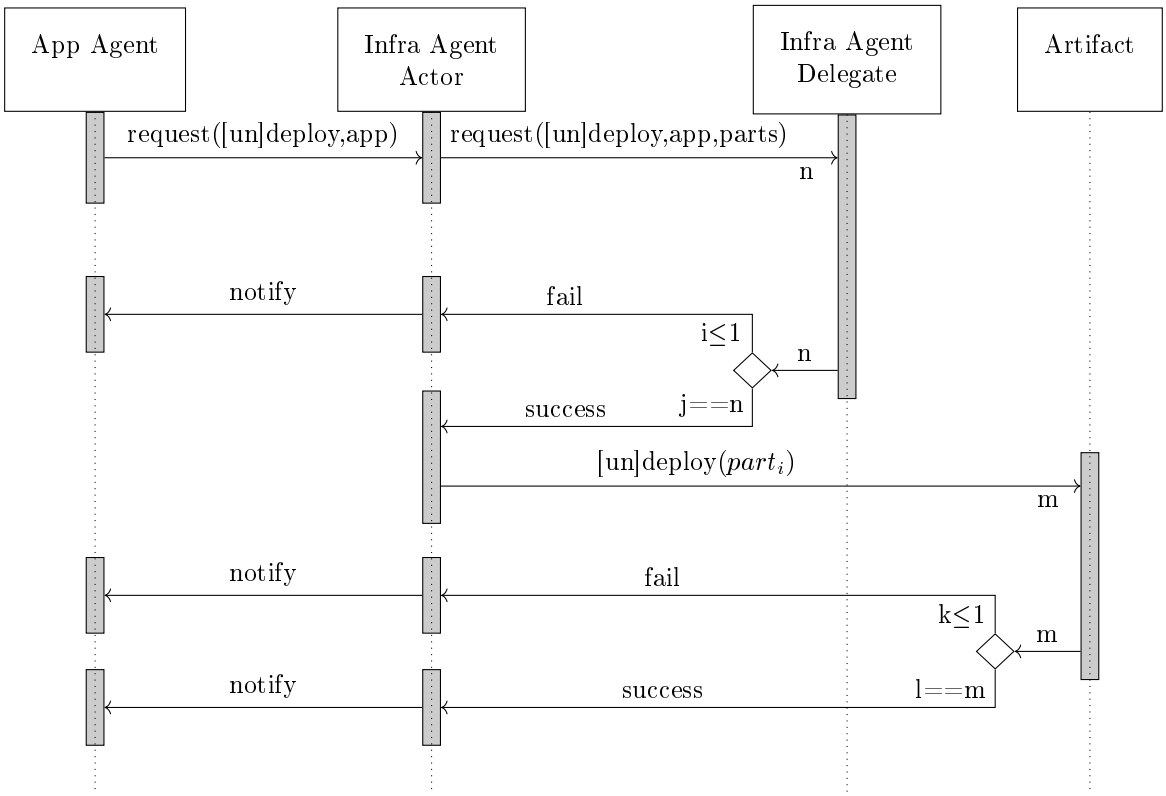


FIGURE 6.7 – Agent Infrastructure : Déploiement d'une partie de l'application

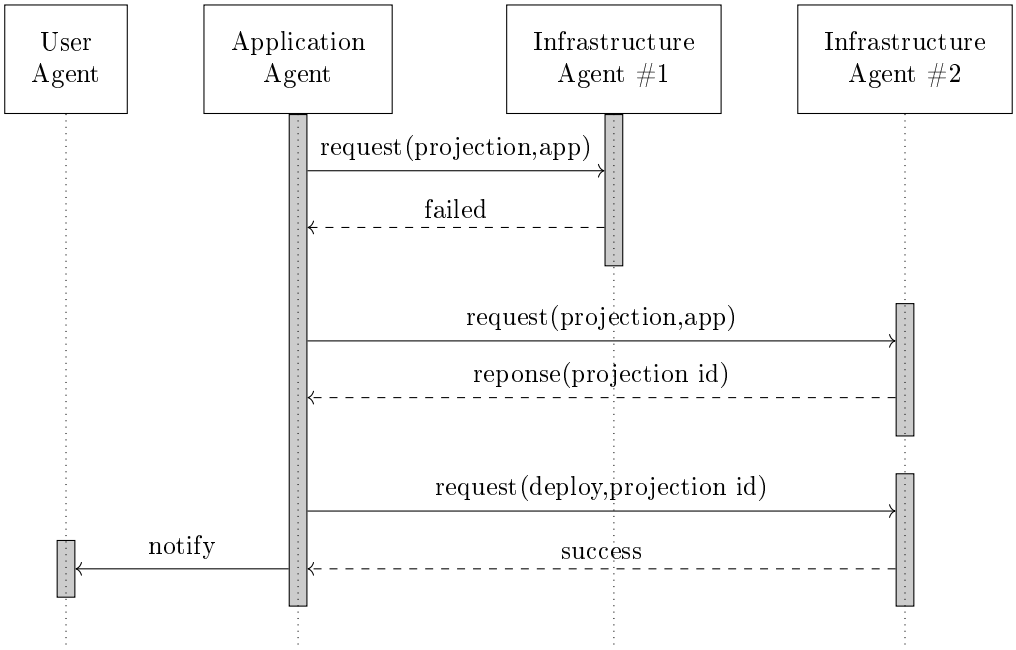


FIGURE 6.8 – Diagramme d'interactions : Exemple déploiement agent application

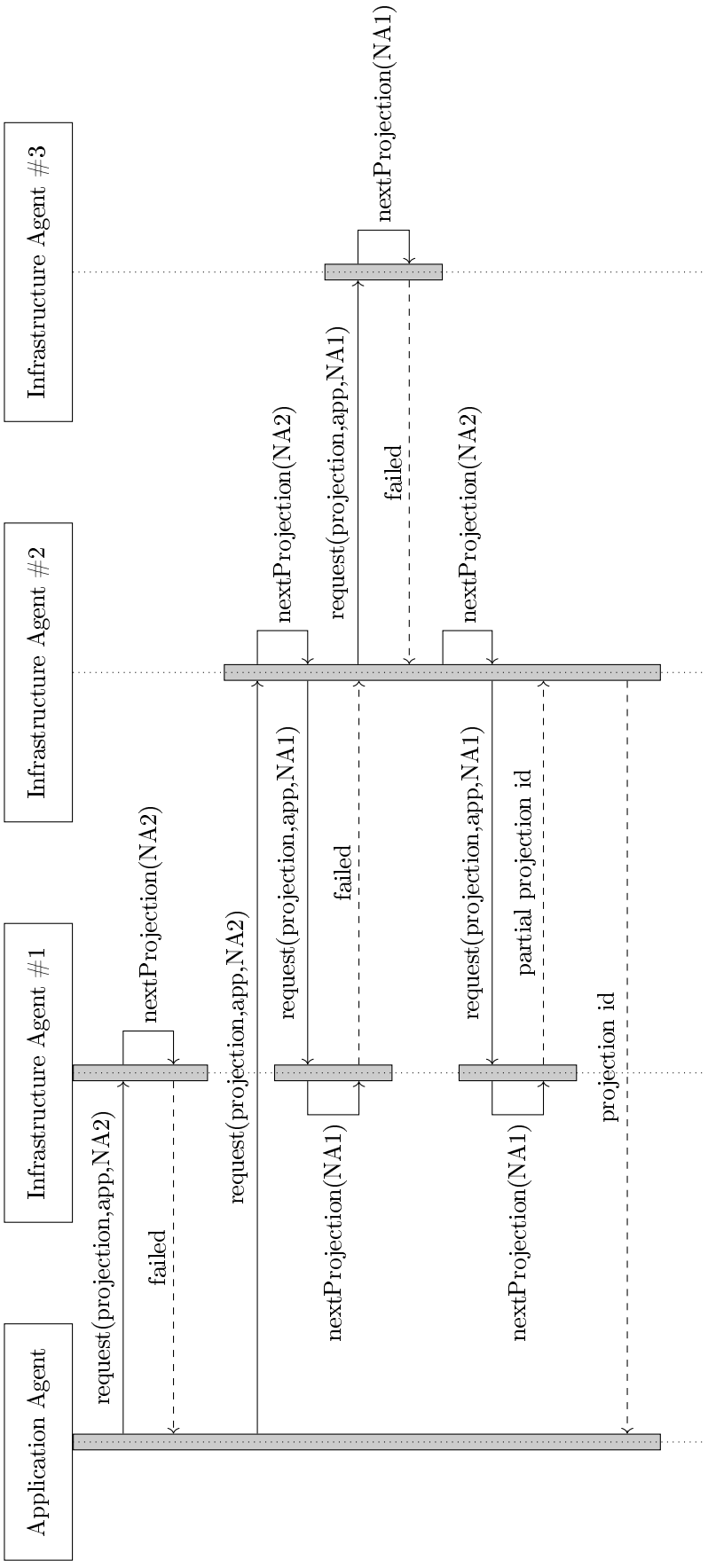


FIGURE 6.9 – Diagramme d'interactions : Exemple projection agent infrastructure

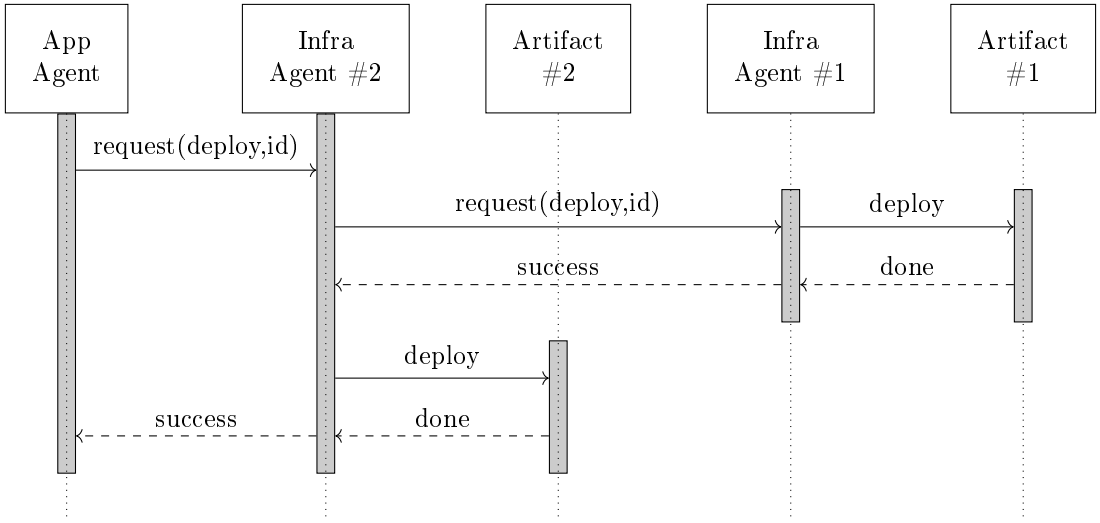


FIGURE 6.10 – Diagramme d'interactions : Exemple déploiement agent application

6.4 Modèle interne des agents

Cette section a pour objectif de détailler le fonctionnement interne des agents. Le comportement des quatre classes d'agents introduites dans la Section 6.2 a été modélisé à l'aide d'une approche orientée but. En effet, une approche à base de buts permet d'augmenter l'autonomie et la proactivité des agents [Cheong 06]. Nous avons modélisé nos agents en suivant l'approche GPS (*Goal Plan Separation*) développée par notre co-auteur Caval et al. [Caval 14]. Dans cette approche, un but est représenté par les plans qui lui sont associés. Un plan ne peut pas à la fois adopter un nouveau but et exécuter une action qui interagit avec l'environnement. Il est ainsi plus facile de dissocier les plan-buts de haut niveau qui décrivent le comportement général de l'agent et les plans d'action de bas niveau qui exécutent les actions concrètes. Chaque agent possède donc un plan-but principal ne contenant pas d'actions et décrivant le comportement haut niveau de l'agent. Ce plan-but principal peut être poursuivi en d'autres plan-buts secondaires, ou directement en plans d'action finaux. Cette approche aide donc à appréhender la complexité de l'agent au travers une description multi-niveaux : du niveau comportemental abstrait à haut niveau, au niveau des actions concrètes à bas niveau. Un autre avantage de cette approche est qu'elle permet de spécifier les relations entre les buts à l'aide de plans.

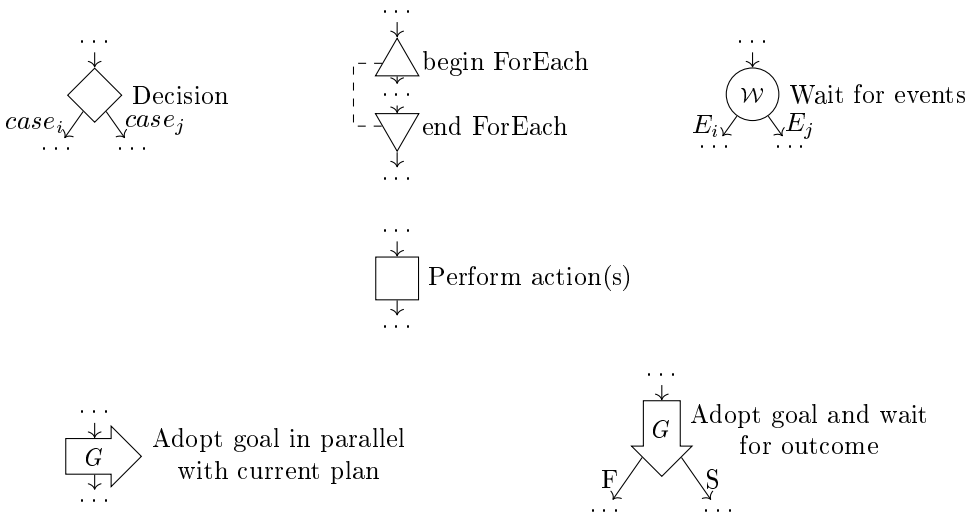


FIGURE 6.11 – Modèle graphique

Ces plans (de but ou d'action) sont représentés à l'aide d'un graphe de flot de contrôle (flowchart) dont les composantes sont détaillées à la Figure 6.11. Cette notation contient les éléments permettant de définir le comportement des agents. Les nœuds de décision permettent d'écrire des tests conditionnels. Les nœuds *ForEach* servent à décrire des boucles et des itérateurs. Les nœuds d'attente (*Wait*) permettent d'introduire la programmation événementielle en déclenchant des actions lors de la perception d'un événement. Les nœuds d'action permettent de déclencher une action : percevoir et interagir avec l'environnement, envoyer des messages à d'autres agents, exécuter l'algorithme de graph-matching, etc. Ces nœuds d'action ne peuvent être utilisés que dans des plans d'action. Enfin, il existe deux types de nœuds permettant une adoption de but et ne pouvant être utilisés que dans des plan-buts. Le premier type permet une

adoption asynchrone d'un but, en parallèle avec l'exécution du plan en cours. Le second type permet une adoption synchrone d'un but. Lors de l'adoption de ce second type, le plan en cours se met en attente jusqu'à ce que le but adopté se termine. Dans ce cas, il est possible de récupérer l'état de retour du but : S si le but a été exécuté avec succès ($End ok$), F si le but a échoué (End_error) ou T si le timeout est levé.

Cette approche permet d'obtenir un cycle de vie des buts relativement simple tout en gardant le bénéfice des caractéristiques d'une approche orientée but classique. La séparation claire entre plan-buts et plan d'action permet une meilleure intelligibilité du comportement des agents.

Dans les sous-sections suivantes, chaque agent sera décrit en utilisant cette approche. Comme le super agent infrastructure ne fait office que de proxy entre les agents infrastructure du groupe et ceux en dehors, son implémentation ne sera pas détaillée ici. Dans ce qui suit, les buts sont écrits sous la forme G_{Xi}, où X correspond au type d'agent et i, au numéro du but. Le plan numéro j correspondant au but G_{Xi} est écrit sous la forme P_{Xi-j}. Dans notre implémentation, un seul plan est associé à chaque but.

6.4.1 Agent infrastructure

Un agent infrastructure gère une partie de l'infrastructure matérielle globale dont il possède une représentation sous la forme d'un graphe présenté dans les chapitres précédents. Cette représentation n'est jamais partagée avec d'autres agents. Cet agent infrastructure est donc le seul à pouvoir raisonner sur ce graphe afin de proposer des solutions pour déployer des applications.

Cette classe d'agent possède différents buts qui sont notamment :
- garder sa représentation de l'infrastructure à jour ;
- proposer des solutions au déploiement d'applications, en considérant les ressources matérielles à sa disposition ainsi que la politique de partage ;
- déployer ou désinstaller les différentes fonctionnalités d'une application en utilisant les artifacts à sa disposition.

Les agents infrastructure répondent donc aux requêtes de projection, de déploiement et de désinstallation des agents application créées par un utilisateur et possédant les bons niveaux d'autorisation.

La Figure 6.12 représente le plan-but de haut niveau adopté au lancement de l'agent infrastructure. Ce plan démarre par l'adoption d'un but de maintenance qui est chargé de maintenir le graphe de l'infrastructure géré par l'agent à jour. En parallèle de l'adoption de ce but, l'agent se met en attente de message en provenance d'autres agents. Lorsqu'il reçoit une requête de déploiement de fonctionnalités, de désinstallation de fonctionnalités ou de calcul d'une projection, il adopte le but correspondant et se remet en attente d'un nouveau message.

Afin de garder à jour la représentation de l'infrastructure matérielle gérée par un agent infrastructure, celui-ci récupère des informations remontées par des artifacts. Ces artifacts renvoient des informations à propos des performances du réseau, de la charge CPU des machines, de l'accessibilité des entités, de l'emplacement géographique d'un

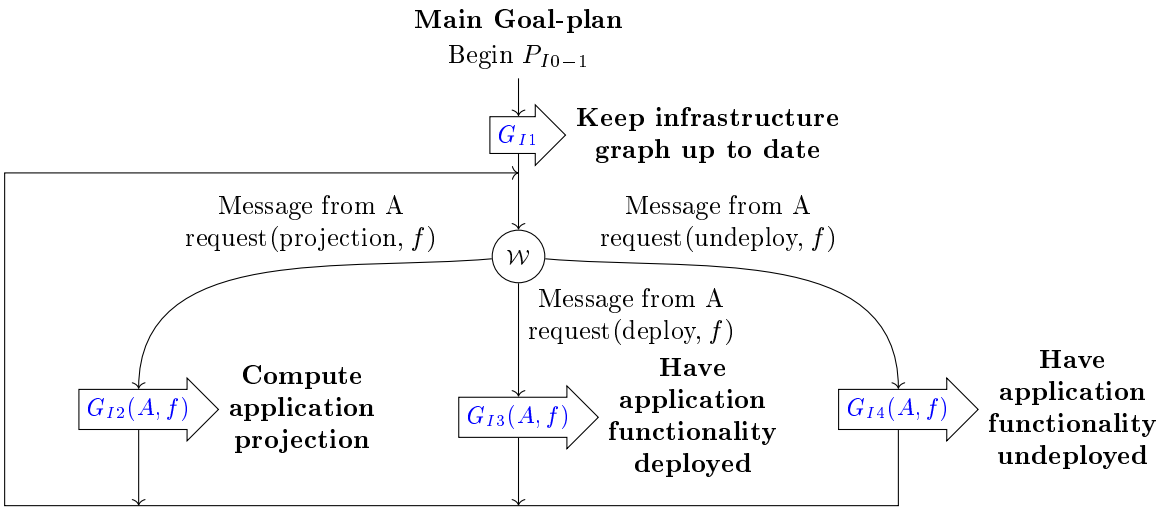


FIGURE 6.12 – Agent Infrastructure : Plan-but principal

utilisateur, ou toute autre information contextuelle. Ceci permet aux agents infrastructure de connaître l'état réel du système. Le but G_{I1}, détaillé par le plan-but P_{I1-1} représenté à la Figure 6.13 permet d'attendre qu'une information extérieure concernant l'environnement lui parvienne. Lorsque c'est le cas, le but $G_{I1.1}$ est alors adopté. Le plan d'action $P_{I1.1-1}$ de ce but est détaillé à la Figure 6.14. Celui-ci met à jour les connaissances de l'agent sur la représentation de l'environnement puis vérifie que toutes les applications utilisant une partie de l'infrastructure prise en charge par l'agent soient consistantes. Si ce n'est pas le cas, l'agent application en question est notifié.

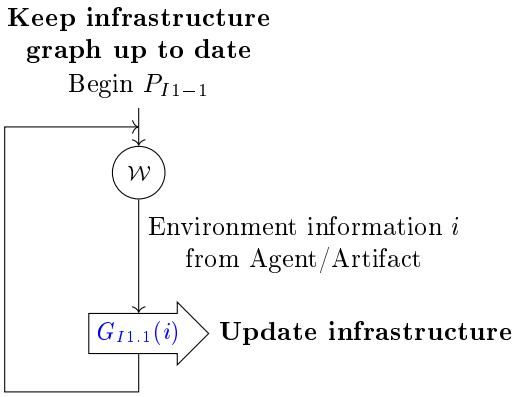


FIGURE 6.13 – Agent Infrastructure Plan pour G_{I1}

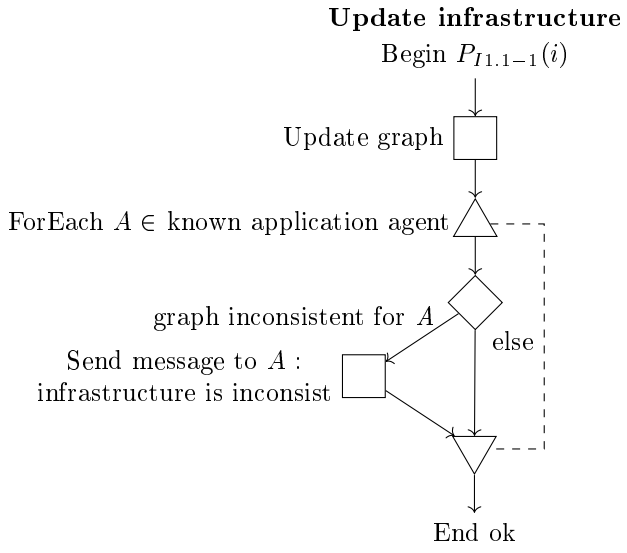


FIGURE 6.14 – Agent Infrastructure Plan pour $G_{I1.1}$

Lorsque l'agent infrastructure reçoit une requête pour déployer une fonctionnalité d'une application, le but G_{I3} est alors adopté. Le plan associé est décrit par la Figure 6.15. Ce plan délègue le déploiement de la fonctionnalité au but $G_{I3.1}$ et attend de manière synchrone le succès ou l'échec du plan associé. Si le but échoue, il se peut que

certaines parties de la fonctionnalité aient déjà été déployées. C'est pourquoi le but de désinstallation de la fonctionnalité $G_{I4.1}$ est adopté. Dans les deux cas, l'agent qui a envoyé une requête de déploiement est notifié du succès ou de l'échec de déploiement.

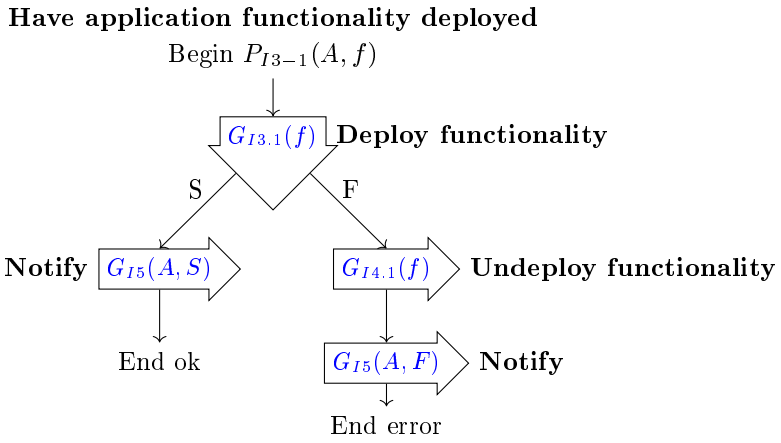


FIGURE 6.15 – Agent Infrastructure : Plan pour G_{I3}

Le déploiement de la fonctionnalité de l'application a été délégué au but $G_{I3.1}$. La Figure 6.16 décrit le plan associé à ce but. Plusieurs actions (ou tâches) peuvent être requises pour déployer la fonctionnalité. Pour chacune de ces tâches, le plan adopte de manière asynchrone le but $G_{I3.1.1}$ qui exécutera celle-ci. Le plan se met ensuite en attente des réponses de ces buts. Si un de ces buts échoue, une des tâches ne s'est donc pas exécutée comme il le fallait et le plan est indiqué comme ayant échoué.

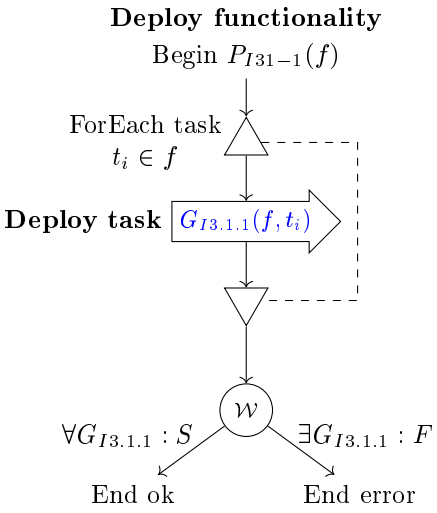


FIGURE 6.16 – Agent Infrastructure Plan pour $G_{I3.1}$

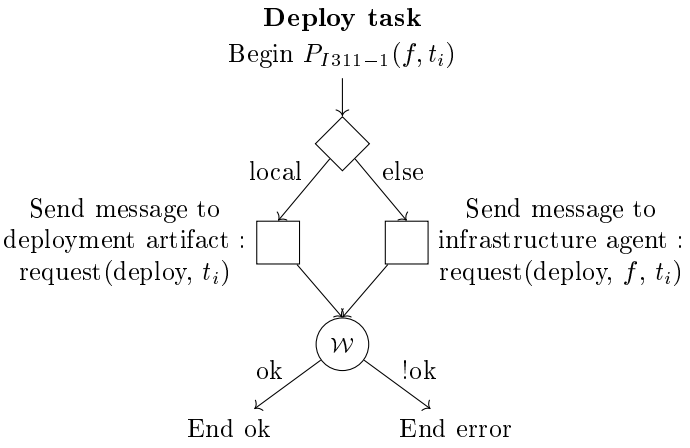


FIGURE 6.17 – Agent Infrastructure Plan pour $G_{I3.1.1}$

La Figure 6.17 décrit le plan qui exécute une des tâches pour le déploiement d'une fonctionnalité. Si cette tâche doit être exécutée par l'agent infrastructure, celui-ci invoque l'artifact de déploiement adéquat. Sinon, il envoie une requête de déploiement à l'agent infrastructure concerné. Lorsqu'une réponse est reçue de l'agent ou de l'artifact, le plan peut se terminer, en échouant ou en réussissant selon que la tâche a pu être exécutée ou non.

Le but G_{I5}, décrit par le plan de la Figure 6.18, permet simplement de notifier un agent A d'un message M.

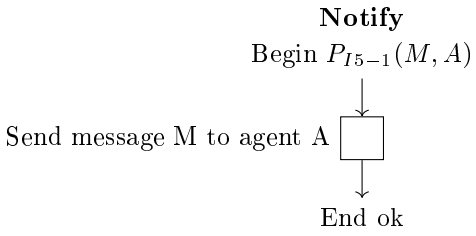


FIGURE 6.18 – Agent Infrastructure : plan pour G_{I5}

Le but G_{I4} pour désinstaller une fonctionnalité d'une application est similaire au but G_{I3} pour le déploiement d'une fonctionnalité. En cas d'échec, l'agent infrastructure va seulement notifier l'agent à l'origine de la requête. La Figure 6.19 détaille le plan correspondant à ce but.

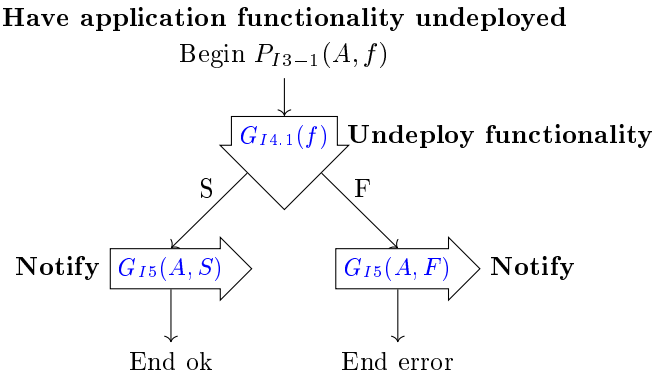


FIGURE 6.19 – Agent Infrastructure : Plan pour G_{I4}

De la même manière, les plans correspondant aux buts $G_{I4.1}$ et $G_{I4.1.1}$ pour la désinstallation d'une fonctionnalité et l'exécution d'une tâche sont similaires aux plans correspondant aux buts $G_{I3.1}$ et $G_{I3.1.1}$. Ceux-ci sont illustrés par les Figures 6.20 et 6.21.

Enfin, le but G_{I2} cherche à calculer une projection d'une application sur l'infrastructure en coopérant avec les agents infrastructure de son voisinage. La Figure 6.22 correspond au plan associé. Une fois le but adopté, l'agent initialise l'algorithme de graph-matching vu au Chapitre 5 et essaye de trouver une solution partielle pour un nœud du super graphe d'application (ce nœud correspond donc à un sous-graphe des besoins de l'application). Si l'algorithme ne fournit pas de projection, l'agent à l'origine de la requête est notifié et le but échoue. Si l'algorithme fournit une projection complète de l'application, alors l'agent à l'origine de la requête est notifié et le but réussi. Enfin, si l'algorithme complète la projection partielle sans réussir à trouver une projection complète, alors une requête est envoyée aux agents infrastructure du voisinage de l'agent courant afin que ceux-ci cherchent à leur tour à compléter la projection de l'application pour les nœuds applicatifs suivants. Le parcours s'arrête si un des agents retourne une solution complète. L'agent essaye de trouver une autre projection si aucun des agents interrogés n'arrive à compléter la projection.

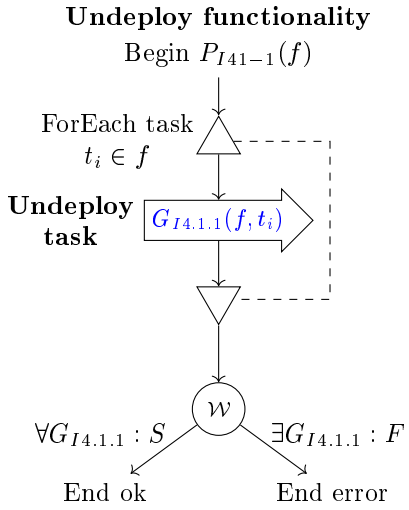


FIGURE 6.20 – Agent Infrastructure Plan pour $G_{I4.1}$

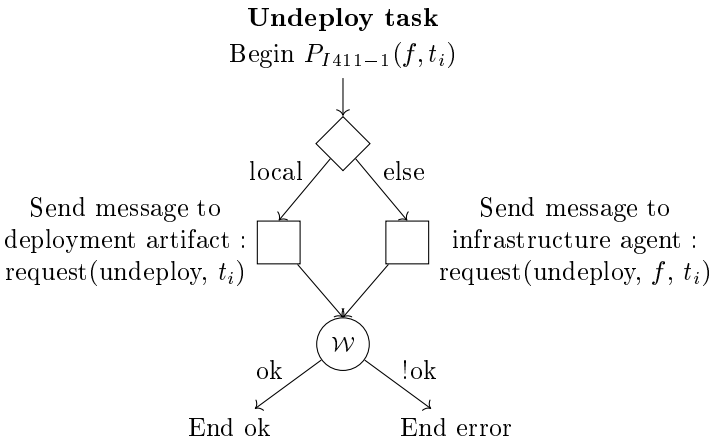


FIGURE 6.21 – Agent Infrastructure Plan pour $G_{I4.1.1}$

6.4.2 Super agent infrastructure

Les agents infrastructure sont organisés et regroupés, formant ainsi des sous-systèmes multi-agents. Afin de faciliter la gestion des politiques de partage, chaque agent d'un même groupe partage par défaut les mêmes niveaux d'autorisation. Ceci permet de définir des groupes privilégiés d'agents. Par exemple, tous les agents du même logement peuvent former un groupe. Ils peuvent ainsi partager entre eux la plupart de leurs ressources. Si un agent extérieur, n'appartenant pas au groupe, réclame l'accès à certaines ressources, alors le niveau de sécurité augmente. Cet agent extérieur, s'il est connu et possède les bonnes autorisations, va donc avoir accès à moins de ressources.

Chaque groupe d'agents infrastructure peut être représenté soit de manière hiérarchique, soit de manière holonique. Une organisation hiérarchique est plus facile à implémenter, mais il faut alors dialoguer avec un agent centralisé par groupe ; notre super agent infrastructure. Cet agent est le représentant de ce cluster et devient l'interface de dialogue privilégiée entre les agents du cluster et l'extérieur de celui-ci. La forme de l'organisation est aussi importante pour fournir de bonnes performances [Fox 81]. Une organisation holonique, introduite pour la première fois par [Koestler 67] dans son livre *The Ghost in the machine*, est quant à elle plus complexe à implémenter, mais permet d'être plus flexible et d'évoluer de manière dynamique. Il n'y a plus un unique agent dédié servant de point d'entrée, mais chacun des agents du groupe peut faire office d'interface entre les autres agents du groupe et l'extérieur. Le super agent infrastructure n'a donc plus de représentation logicielle et tout agent du groupe fait office de représentant de celui-ci, ce qui augmente la robustesse du système, l'agent dédié dans une organisation hiérarchique étant considéré comme un *single point of failure*. Néanmoins, les organisations hiérarchiques et holoniques ne changent pas en profondeur la manière dont sera utilisé le système multi-agents. En effet, certaines hiérarchies sophistiquées se rapprochent même très fortement d'une organisation holonique [Horling 04].

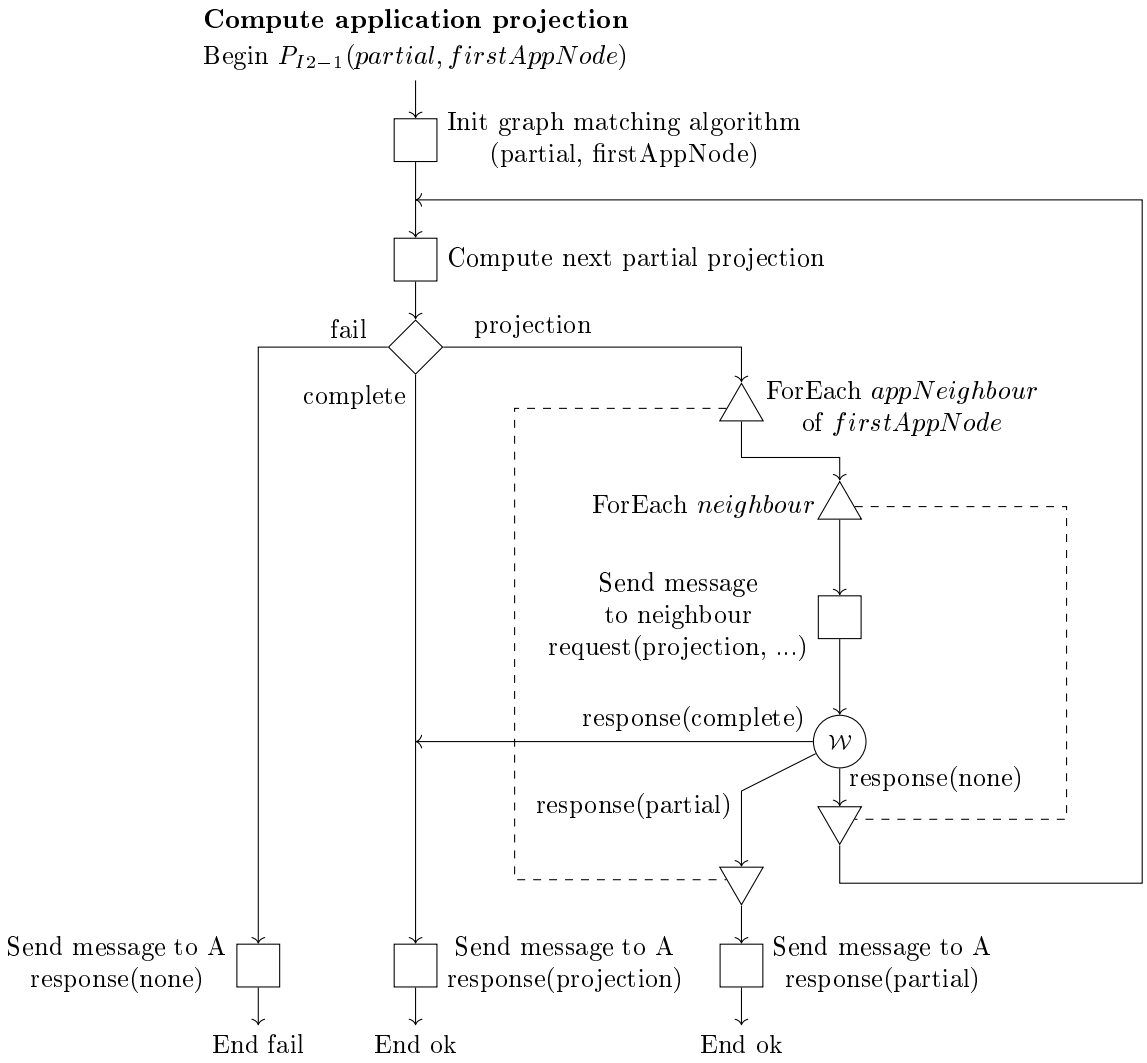


FIGURE 6.22 – Agent Infrastructure : plan pour G_{I2}

Nous avons choisi d'implémenter une organisation hiérarchique, d'une part pour sa simplicité et d'autre part car le super agent infrastructure, faisant l'interface entre les agents du groupe et les agents extérieurs, permet d'abstraire et de cacher les agents du groupe vis-à-vis de l'extérieur. Ceci contribue à augmenter le niveau de confidentialité des ressources. Un super agent infrastructure peut lui-aussi appartenir à un groupe d'agents infrastructure. Il est ainsi possible de former une organisation multi-échelle en cachant à plusieurs niveaux des groupes d'agents infrastructure ou de super agents infrastructure. Les agents présents dans une maison forment un cluster logement, différents clusters logements peuvent former un cluster bâtiment, et ainsi de suite.

6.4.3 Agent application

Un agent application est créé par un agent utilisateur lorsque son utilisateur souhaite déployer une nouvelle application. Cet agent gère la vie d'une application, de son déploiement à sa désinstallation, en garantissant sa consistance tout au long de son

exécution. Cette classe d'agent possède la description de l'application sous la forme d'un graphe qui décrit les différentes fonctionnalités de l'application ainsi que leurs besoins matériels pour être déployées correctement. Ces fonctionnalités peuvent être déployées sur différentes parties de l'infrastructure, gérées par différents (super) agents infrastructure. De ce fait, l'agent application doit interagir avec ces différents agents infrastructure dans le but de déployer ou de désinstaller son application.

Cet agent application a pour but de :
— Coopérer avec les agents infrastructure pour déployer l'application ou des parties de cette application
— Demander la désinstallation de parties de l'application
— Maintenir la cohérence de l'application et redéployer certaines parties si besoin
Un agent application possède les mêmes niveaux d'autorisation que l'agent utilisateur qui l'a créé. Il peut ainsi envoyer des requêtes aux agents infrastructure qu'il connaît.

La Figure 6.23 représente le plan-but principal, qui est exécuté au lancement de l'agent. Ce plan-but commence par l'adoption, de manière asynchrone, d'un but G_{A1} visant à déployer une partie de l'application. En parallèle du déploiement initial, l'agent se met en attente de message ou d'événement interne. Lorsqu'il reçoit un message d'un agent infrastructure informant qu'une partie de l'application déployée est inconsistante, l'agent redéploie la partie en question au travers du but G_A3. Lorsque l'agent reçoit un événement interne de déploiement ou de désinstallation d'une partie de l'application, celui-ci adopte le but adéquate (G_{A1} ou G_{A2}).

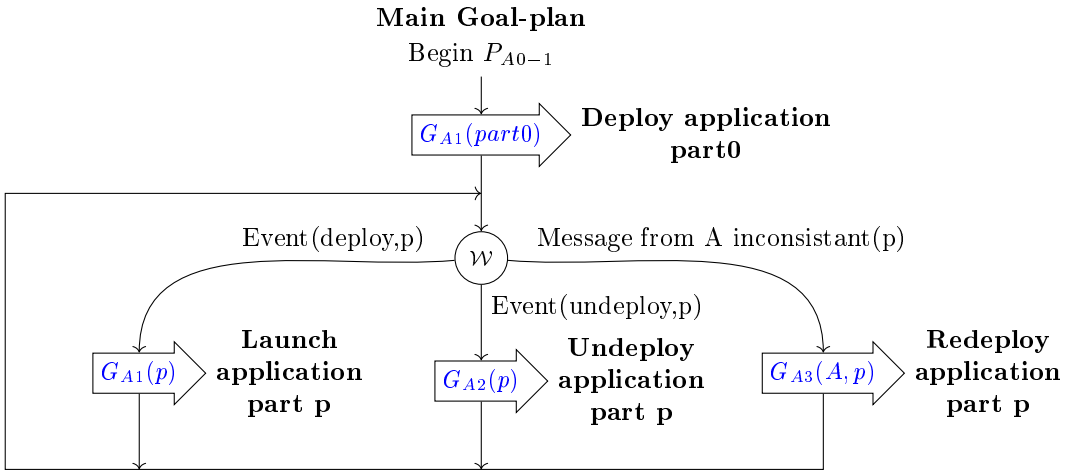


FIGURE 6.23 – Agent Application : Plan-but principal

La Figure 6.24 détaille le fonctionnement du plan correspondant au but de redéploiement d'une partie de l'application G_{A3}. Ce plan va simplement désinstaller la partie de l'application qui est déjà en cours d'exécution en adoptant de manière synchrone le but G_{A2} puis va lancer le but G_{A1} pour déployer de nouveau cette partie.

Le plan correspondant au but G_{A1} de déploiement d'application est décrit à la Figure 6.25. Ce plan-but essaye d'obtenir une projection d'une partie de l'application ($G_{A1.1}$) avant de demander le déploiement effectif de celle-ci ($G_{A1.2}$) Si une projection est trouvée, mais que le déploiement échoue, alors l'agent réadopte le but $G_{A1.1}$ afin de trouver une autre projection.

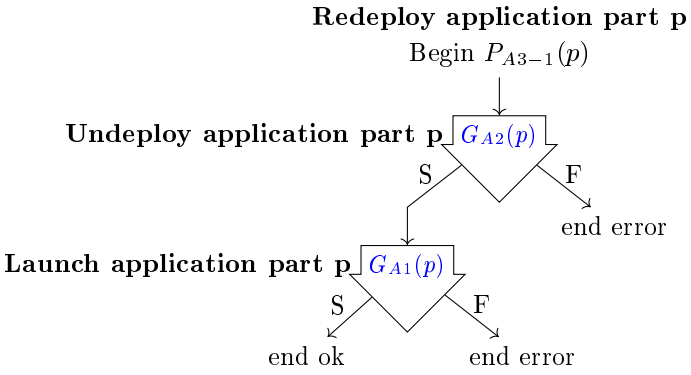


FIGURE 6.24 – Agent Application : plan pour G_{A3}

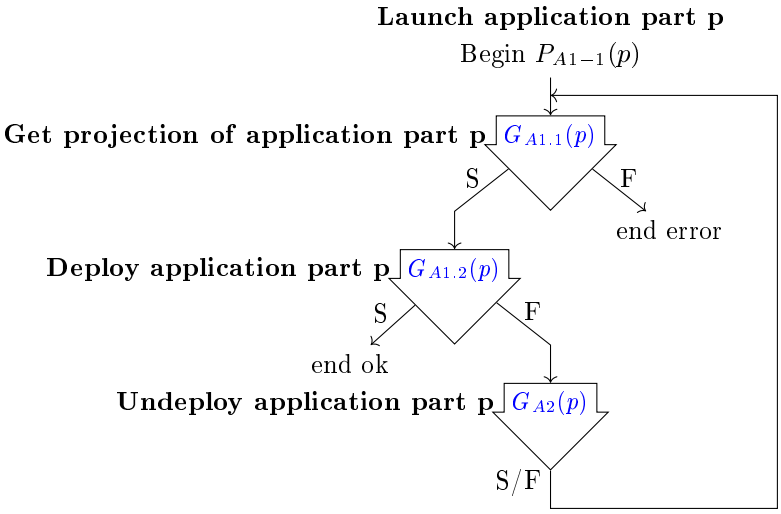


FIGURE 6.25 – Agent Application : plan pour G_{A1}

Le but $G_{A1.1}$ vise à obtenir une projection d'une partie de l'application. Pour cela, le plan correspondant (Figure 6.26) va interroger de manière synchrone tous les agents infrastructure pour lesquels l'agent application possède des autorisations. Si un de ces agents infrastructure fournit une projection valide, le plan se termine avec succès. Si aucun des agents n'est capable de fournir une projection, le plan échoue.

Le déploiement d'une partie de l'application ($G_{A1.2}$), ou sa désinstallation (G_{A2}) sont des processus similaires. Les Figures 6.27 et 6.28 correspondent aux plans de ces deux buts. L'agent application va, pour chaque fonctionnalité de la partie d'application à déployer ou à désinstaller, demander à l'agent infrastructure correspondant un déploiement ou une désinstallation de celle-ci. Ces demandes sont envoyées de manière asynchrone. Le plan se met ensuite en attente d'une réponse de tous les agents infrastructure afin de pourvoir terminer le but, ou de le faire échouer.

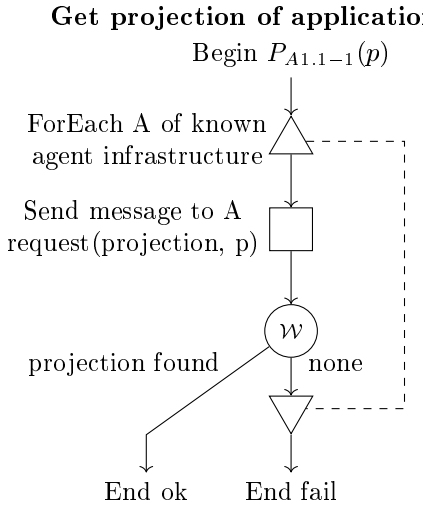


FIGURE 6.26 – Agent Application : plan pour $G_{A1.1}$

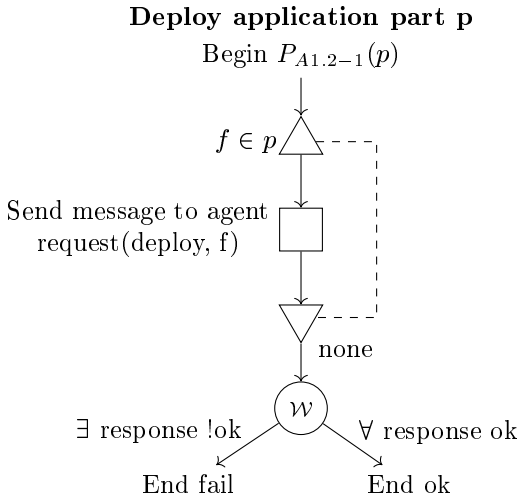


FIGURE 6.27 – Agent Application Plan pour $G_{A1.2}$

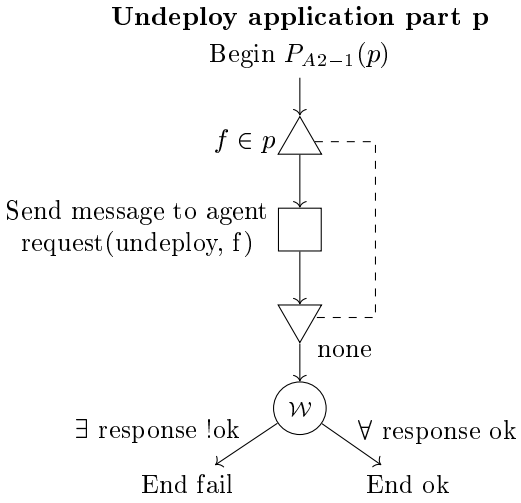


FIGURE 6.28 – Agent Application Plan pour G_{A2}

6.4.4 Agent utilisateur

Un agent utilisateur est l'interface privilégiée entre l'utilisateur et le système multi-agents global. Un agent utilisateur est associé à un unique utilisateur et garde en mémoire ses préférences, ses habitudes, sa position, etc. Cet agent est conscient du contexte associé à l'utilisateur. Il autorise celui-ci à explorer le store d'applications disponibles, choisir une d'entre elles et demander de la déployer et de l'activer. L'agent utilisateur va alors venir créer un agent application qui va récupérer le graphe de l'application associée et va débuter une coopération avec des agents de déploiement, en fonction des préférences de l'utilisateur et du contexte, afin de trouver une solution.

Les buts de l'agent sont donc de :
— Déployer une nouvelle application
— Désinstaller une application

— Vérifier que les agents application créés soient fonctionnels

Un agent utilisateur peut être attaché à certains super agents infrastructure présentés dans la section précédente, avec différents niveaux d'autorisation. Par défaut, tout agent utilisateur non reconnu par le groupe d'agents n'a pas le droit d'utiliser les ressources matérielles gérées par ce groupe.

La Figure 6.29 représente le plan-but principal de l'agent utilisateur. Ce plan-but se met en attente d'une requête de l'utilisateur et adopte le but adéquat pour la création d'une nouvelle application G_{U1} ou pour désinstaller une application existante G_{U2}

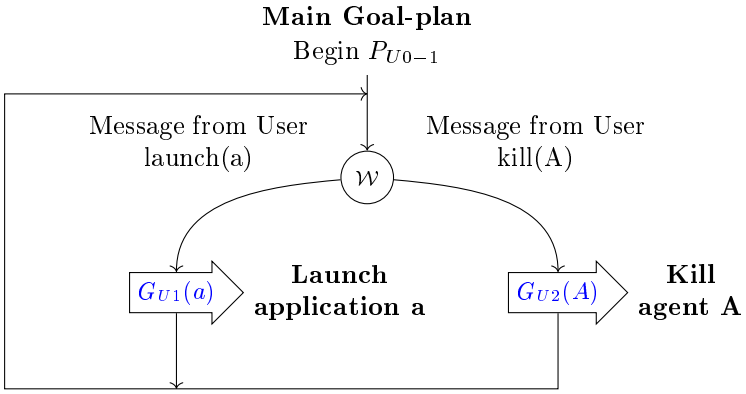


FIGURE 6.29 – Agent Utilisateur : plan-but principal.

Lors de la création d'une nouvelle application, le plan décrit à la Figure 6.30 est exécuté. Celui-ci essaye de créer un nouvel agent application en adoptant le but $G_{U1.1}$, représenté par la Figure 6.31. Dès qu'il a réussi, il adopte le but de maintenance de cet agent $G_{U1.2}$ décrit à la Figure 6.32.

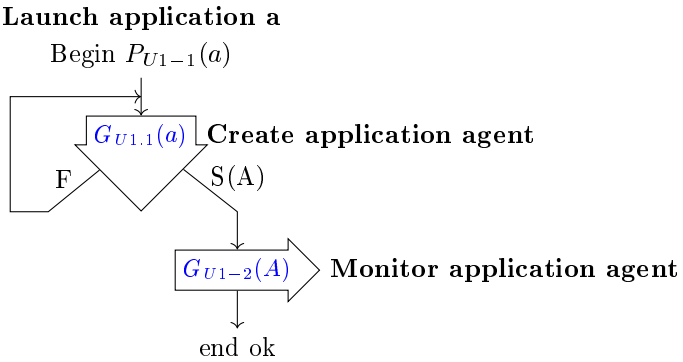


FIGURE 6.30 – Agent Utilisateur : plan pour G_{U1}

Enfin, lorsque l'utilisateur souhaite désinstaller une application, le plan correspondant au but G_{U2} est exécuté (Figure 6.33). Ce plan envoie un message à l'agent application pour que celui-ci désinstalle ce qui est déployé. Que la réponse soit positive, négative ou qu'il n'y en ait pas, l'agent utilisateur se charge de tuer l'agent application avant de terminer le plan.

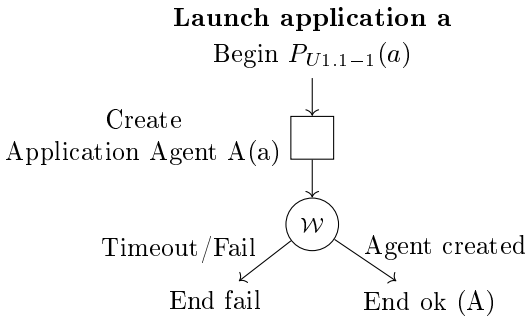


FIGURE 6.31 – Agent Utilisateur Plan pour $G_{U1.1}$

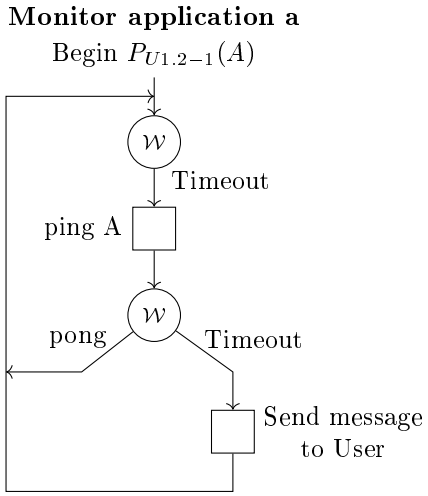


FIGURE 6.32 – Agent Utilisateur Plan pour $G_{U1.2}$

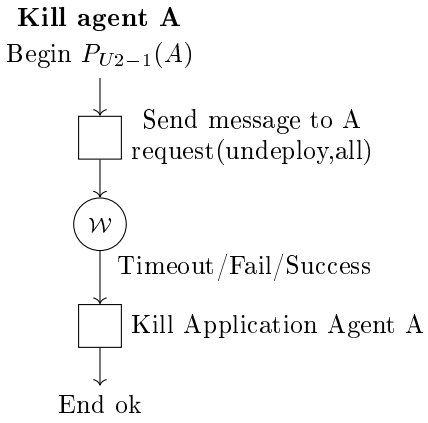


FIGURE 6.33 – Agent Utilisateur : plan pour G_{U2}

6.5 Conclusion

Dans ce chapitre, nous avons proposé une approche multi-agents pour répondre aux problèmes de la dynamicité des environnements ainsi que de la confidentialité des ressources de l'infrastructure.

Après avoir extrait du scénario présenté au Chapitre 3.1 les besoins et les différents rôles du système, nous avons introduit les quatre classes d'agents composant le SMA. Les interactions entre ces classes d'agents ont été établies et le comportement interne de chaque agent a été détaillé à l'aide d'une approche orientée but. Ce quatre classes d'agents permettent une séparation claire entre l'utilisateur, les applications et l'infrastructure matérielle. Afin de préserver la confidentialité des ressources, les graphes décrivant l'infrastructure matérielle sont pris en charge localement par les agents infrastructure concernés. Ces agents sont organisés de manière hiérarchique afin de créer une organisation multi-échelle. La projection des besoins matériels des applications sur l'infrastructure est réalisée à l'aide de la version distribuée de l'algorithme de graph-

matching présentée au Chapitre 5. Afin d'interagir avec l'environnement réel, deux classes d'artifacts ont aussi été définies.

L'utilisation d'un système multi-agents a rendu possible l'introduction de mécanismes renforçant la confidentialité des ressources au niveau de l'architecture et de l'organisation du SMA. En plus de cela, nous avons introduit des politiques de partage afin de contrôler l'utilisation et le partage des ressources. Ceci a été un critère important pour le choix du paradigme agent. En effet, dans le domaine de l'Intelligence Ambiante, l'infrastructure matérielle est souvent possédée par différents acteurs qui ont besoin de garantir la confidentialité de leurs ressources. La séparation entre les couches applicative et infrastructure, couplée à une approche décentralisée, augmente la robustesse de la solution. L'utilisation d'une représentation orientée but, avec l'utilisation de l'approche GPS (Goal Plan Separation) a facilité la modélisation des différentes tâches des agents.

Troisième partie

Expérimentations

Chapitre 7

Expérimentations

Sommaire

Nous avons présenté dans la seconde partie de ce manuscrit les différentes contributions au problème du déploiement d'applications dans un environnement ambiant. Le Chapitre 3 a introduit un métamodèle pour décrire l'infrastructure matérielle ainsi que les applications de tels environnements. Le Chapitre 4 a offert une formalisation mathématique du déploiement. En trouvant un homomorphisme entre le graphe d'infrastructure et le patron de l'application à déployer, il est possible de sélectionner les entités de l'infrastructure qui supporteront le déploiement de l'application. Le Chapitre 5 a proposé des algorithmes de graph-matching, notamment une version distribuée permettant d'améliorer la confidentialité des informations relatives à l'infrastructure, pour trouver un de ces homomorphismes. Enfin, le Chapitre 6 a permis d'encapsuler ces mécanismes au sein d'un système multi-agents améliorant la scalabilité et fournissant des mécanismes pour préserver la confidentialité des ressources des utilisateurs. Dans ce dernier chapitre, nous proposons de mettre en œuvre ces apports théoriques au sein d'un intergiciel pour le déploiement d'application dans un système ambiant. La Section 7.1 introduit le bâtiment Urbawood dans lequel se trouve une réplique de maison intelligente que j'ai eu la chance de pouvoir utiliser dans le cadre de mes recherches. Nous présenterons une ébauche de l'intergiciel dans la Section 7.2 en détaillant les choix de conception. Enfin, nous proposerons des résultats permettant d'évaluer l'algorithme de projection à la Section 7.3.

7.1 Laboratoire d'Intelligence Ambiante de l'ISEN

J'ai effectué mes travaux de recherche au sein du Laboratoire d'Intelligence Ambiante de l'ISEN à Lille. J'ai notamment eu accès aux nouvelles installations du laboratoire, à savoir une réplique de maison intelligente, entièrement fonctionnelle, basée à Urbawood, sur le site d'EuraTechnologies. Ce démonstrateur a pour objectif de permettre la mise en œuvre d'objets connectés dans un environnement réel afin d'étudier les possibilités offertes par ceux-ci, d'imaginer des scénarios pour la maison du futur et de proposer des solutions permettant de les faire interagir de manière intuitive et transparente pour l'utilisateur. Les objectifs du laboratoire en terme de recherche se focalisent sur deux aspects. Le premier, qui correspond à celui traité dans cette thèse, est celui de l'intégration et l'utilisation des objets connectés dans l'environnement en privilégiant des interactions horizontales. Le second aspect concerne la proposition d'interfaces multi-modales intuitives permettant à l'utilisateur d'interagir de manière transparente avec son environnement.

Une partie de mon travail a consisté à étudier et à tester différents types d'objets connectés achetés dans le commerce. Nous avons ainsi pu nous rendre compte de l'hétérogénéité des technologies et des protocoles utilisés. Entre les caméras IP, l'interrupteur EnOcean, l'alarme ZWave et les différents types d'écran (tactile, table basse, télévision, miroir connecté, etc.), il est rapide de se rendre compte des difficultés à faire interagir ces objets. De plus, certains objets n'autorisent même pas l'utilisateur à récupérer directement les données capturées. La balance connectée de la marque Withings par exemple est construite autour d'une architecture complètement fermée qui ne permet pas la récupération directe des données produites par la balance. Ces dernières sont envoyées aux serveurs de la société qui nous fournit en retour des outils et

des services de visualisation de la courbe de poids. Même arrivé à ce stade et jusqu'à il n'y a récemment, aucune API n'était fournie pour récupérer ces données stockées chez le fournisseur. Pour les récupérer directement à partir de la balance sans passer par les serveurs externes, il a fallu intercepter le flux de données envoyé par la balance et l'interpréter nous-même. Fort heureusement (ou plutôt aussi incroyable que cela puisse paraître), les données envoyées sur l'Internet ne sont pas cryptées (c'est le cas pour 90% des objets connectés commercialisés [HP 14]). Un autre produit de la société Sen.se (société qui a, entre autre, commercialisé le lapin Nabaztag), appelé Mother, est construit sur un modèle communautaire. Le produit fournit un ensemble de « cookies » qui regroupent un certain nombre de capteurs (capteur de température, accéléromètre, etc.) ainsi qu'une centrale qui récupère les données produites par ces cookies pour les utiliser ou les envoyer sur l'Internet. Tout l'intérêt de ce produit est que la société propose un store d'applications qui utilisent ces données. Mother peut alors se transformer en compteur de cafés si le cookie est collé sur la machine à café, en détecteur d'incendie grâce à son capteur de température, ou encore en notificateur d'ouverture de porte. Malheureusement aucun SDK n'est fourni et il est impossible pour un utilisateur lambda d'imaginer, de créer et de partager ses propres scénarios d'utilisation du produit. Enfin, pour donner un dernier exemple des nombreux produits testés, les caméras IP mettent à disposition de l'utilisateur des services web permettant de configurer automatiquement la caméra ainsi que de récupérer par plusieurs clients en même temps le flux vidéo généré. Ce type de produit est le plus ouvert de tous et le plus facile à intégrer dans un environnement ambiant.

FIGURE 7.1 – Urbawood : réplique fonctionnelle d'une maison intelligente

Une fois le fonctionnement de ces objets cerné, nous avons pu en équiper la maison intelligente afin d'implémenter différents scénarios impliquant l'utilisation conjointe de

ces objets. Moyennant quelques adaptations pour certains produits afin de pouvoir récupérer directement les données produites, il nous a été possible de faire interagir ces objets. Des interfaces multi-modales (plan 3D du bâtiment à la Figure 7.2) ont été développées afin de pouvoir contrôler les différents capteurs sur différents types de terminaux (télévision, miroir connecté, table basse tactile). Les différents scénarios

FIGURE 7.2 – Contrôle de l'environnement via la table basse tactile

présentés au Chapitre 3.1 ont notamment été implémentés. Ainsi, un système de localisation à l'aide de Beacons bluetooth a été mis en place, permettant au portier vidéo d'afficher l'image de la caméra IP d'entrée sur l'écran adapté : sur la télévision lorsque l'utilisateur se trouve dans le salon ou sur le miroir connecté quand ce dernier rentre dans la salle de bain. La télévision a été implémentée par une application web qui peut être déployée sur un écran et peut être commandée à l'aide de plusieurs interfaces : soit avec la traditionnelle télécommande, mais aussi grâce à une interface dédiée apparaissant sur la table basse tactile du salon, ou alors avec un système de jeu de cartes dont les symboles sont reconnus par la caméra de l'appartement et permettent d'effectuer diverses actions. D'autres scénarios et applications ont été mis en œuvre. Ainsi par exemple, le plan de travail de la cuisine s'adapte automatiquement à la taille de la personne voulant l'utiliser. La consommation énergétique de la maison a été monitorée. La fréquence de récupération des données a été augmentée au maximum afin de pouvoir suivre en temps réel la consommation des utilisateurs. Cela nous permet d'extraire des informations sur le mode de vie des usagés et par la même occasion de soulever des questions sur la confidentialité des informations – même la récupération de simples données énergétiques peut nuire à la confidentialité des habitudes des usagés. Cette approche pratique nous a permis de réaliser les difficultés de faire communiquer des entités non directement interopérables et utilisant des technologies hétérogènes.

7.2 Intergiciel pour le déploiement d'applications

7.2.1 Principe général

L'intergiciel vise à fournir une implémentation des différents outils et mécanismes présentés dans les chapitres précédents afin de déployer automatiquement un certain nombre d'applications au sein d'un environnement ambiant. Cet intergiciel se divise en quatre grandes parties :

— La partie agents
— La partie algorithme de projection
— La partie graphe et ontologie
— La partie artifacts de monitoring et de déploiement
— La partie IHM et monitoring

La partie agents offre une première implémentation des différents types d'agents, de leurs protocoles d'interaction présentés au Chapitre 6, ainsi que des politiques de confidentialité. La partie algorithme de projection propose une implémentation générique de l'algorithme distribué permettant de trouver le plan de déploiement d'une application sur une infrastructure. La partie graphe et ontologie propose des modules de gestion de graphes et de gestion de l'ontologie proposée au Chapitre 3. Cette ontologie encode en particulier les différents concepts ainsi que les différents opérateurs relatifs aux propriétés des nœuds. Cette ontologie est connue de tous les agents et est utilisée plus spécifiquement par les agents infrastructure. La partie artifacts offre une interface générique pour dialoguer avec des artifacts de déploiement qui pourront être développés utilisant n'importe quelle technologie, pourvu que l'interface permettant de manipuler ces entités soit respectée. Enfin, la partie IHM et monitoring offre tout un ensemble d'outils graphiques permettant de superviser le système et d'interagir avec. Ces interfaces permettent notamment de visualiser l'état des agents du système, de les déboguer, d'éditer les graphes d'infrastructure et des applications, de visualiser les applications déployées dans l'environnement, etc.

Dans la version actuelle, les parties agent, algorithme, graphe et ontologie ont été développées, ce qui nous a permis de tester les performances de la version distribuée de l'algorithme ainsi que la coopération entre les agents. Le reste est en cours de développement. Une interface basique avec les artifacts de déploiement est proposée et une implémentation de deux agents de déploiement (SSH et puppet) permet de valider le concept. Les différentes interfaces ne sont pas encore développées et tout se fait pour le moment en ligne de commande.

7.2.2 Implémentation

Un des besoins pour l'implémentation du système multi-agents est que les agents puissent être exécutés indépendamment, sur les périphériques présents au sein d'un logement tels qu'une box télévision ou domotique, un smartphone, ou un micro-ordinateur. La puissance de ces machines étant limitée, il nous fallait une plateforme légère et facilement déployable sur ces types de périphériques. De plus, les technologies web prennent de plus en plus de place dans le monde du développement logiciel actuellement. Ceci

est dû, d'une part, à l'évolution rapide des langages, à leur permissivité et expressivité et d'autre part, à leur totale interopérabilité. Il est maintenant possible de développer facilement toutes les facettes d'une application (serveur/api, client/IHM) en utilisant les technologies web et en particulier des langages tels que Javascript qui offrent des mécanismes puissants tels que l'introspection, une interface de métaprogrammation, les closures, l'héritage prototypale, les lambdas, la manipulation facile d'itérateurs grâce aux générateurs et bien plus encore. Ces applications peuvent fonctionner sur tous types de micro-ordinateurs, même avec peu de ressources.

La plupart des plateformes multi-agents existantes, comme Jade, consomment énormément de mémoire et sont développées en Java [Kravari 15]. Elles offrent certes un certain nombre de mécanismes puissants, mais sont malheureusement trop lourdes et ne remplissent pas nos critères. C'est pourquoi nous avons choisi de redévelopper notre propre plateforme multi-agents en utilisant les technologies web et en se basant sur la bibliothèque d'agents Evejs. Evejs est une plateforme orientée agents minimaliste, développée en javascript, qui met à disposition du développeur des coquilles vides pouvant communiquer entre elles à l'aide de nombreux protocoles et couches de transport existants. Ce sont ces coquilles qui servent de base à nos agents.

Les développements ont été effectués en utilisant la machine virtuelle V8 encapsulée dans la plateforme NodeJS[1]. Cette plateforme possède l'avantage de mettre à disposition un bon nombre de bibliothèques (au travers du gestionnaire de paquet NPM[2]) permettant d'interagir de manière générique avec le système sur lequel elle s'exécute. Le code source est écrit en javascript selon la dernière norme ECMAScript ES7[3] afin de bénéficier des dernières spécificités du langage. Ce code est ensuite transpilé automatiquement dans la norme ES5 afin d'être compatible à 100% avec la machine virtuelle V8.

7.2.2.1 Algorithme de projection

La Figure 7.3 représente le diagramme de classes UML simplifié de la partie gérant l'algorithme de projection, la représentation graphe et l'ontologie. Cette partie est centrée autour de la classe GraphMatching qui calcule les différentes projections du pattern d'application sur le graphe de l'infrastructure selon le mécanisme sélectionné et retourne des instances de la classe Projection qui correspondent aux projections complètes ou partielles. Les classes BasicMechanism et SuperGraphMechanism qui héritent de la classe abstraite Mechanism, implémentent les itérateurs de parcours respectivement pour l'algorithme centralisé et pour sa version distribuée. Cela permet à la classe GraphMatching d'être générique et de proposer les mécanismes globaux de parcours et de backtrack sans se soucier de la nature des graphes à projeter (patron d'application sur infrastructure, ou super graphe du patron et de l'infrastructure). L'on remarquera que la classe SuperGraphMechanism a besoin d'utiliser l'algorithme et donc la classe GraphMatching afin de déterminer les projections partielles pour un sous-graphe du patron de l'application. Les classes SolutionStack et ExplorationStack héritent toutes

1. https://nodejs.org/en/
2. Node Package Manager
3. http://www.ecma-international.org/ecma-262/7.0/index.html

deux de la classe Stack et offrent, en plus des mécanismes génériques de manipulation des éléments d'une pile, des mécanismes spécifiques permettant de rechercher dans ses éléments selon certains critères. Ainsi, la fonction $getFibers$ de la classe SolutionStack par exemple, prend en paramètre l'identifiant d'un nœud du graphe de l'infrastructure et retourne la liste des éléments de la pile contenant les nœuds du patron de l'application qui ont été associés au nœud infrastructure. La classe Ontology possède les informations sur les opérateurs des propriétés des nœuds des graphes et offre des fonctions permettant de combiner et de comparer ces propriétés. Cette classe est utilisée par l'algorithme lui-même. Enfin, la classe RandomGraphGenerator est une classe utilitaire permettant de générer aléatoirement, grâce à un ensemble de paramètres, un graphe orienté, basé sur l'ontologie définie par la classe Ontologie.

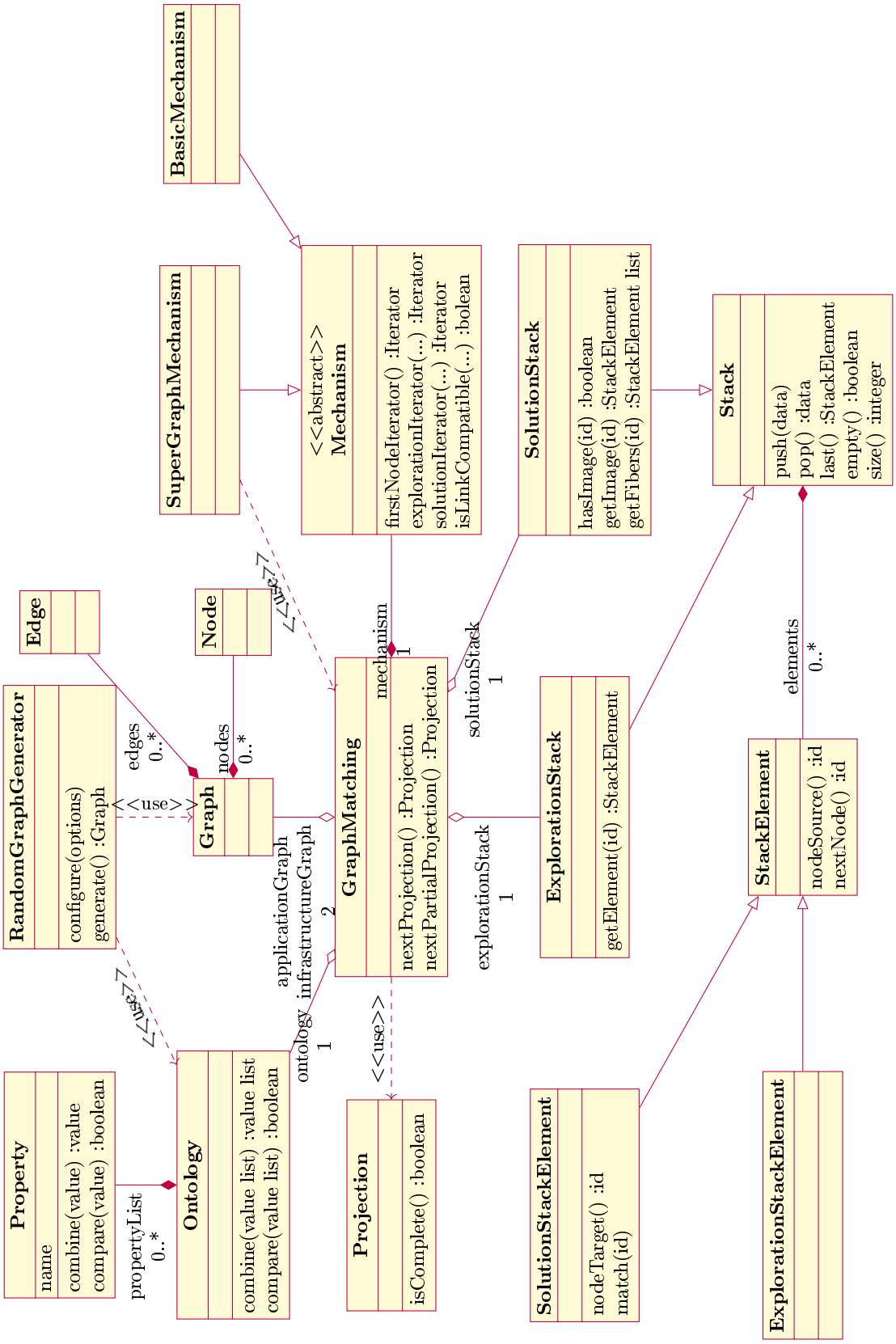


FIGURE 7.3 – Diagramme de classes UML de la partie algorithmique

7.2.2.2 Agents et interfaces Hommes/Machines

Figure 7.4 représente le diagramme de classe UML simplifié de la partie gérant les agents, les artifacts ainsi que les interfaces hommes-machines. Un agent est basé sur la classe eve.Agent du framework evejs fournissant un ensemble de protocoles et de couches de transport permettant aux agents de s'envoyer des messages. La classe Agent réutilise ces mécanismes de communication et réimplémente deux patrons de communication afin de faciliter les échanges de messages entre les agents. Le premier patron de communication implémenté est le patron Request/Response permettant à un agent d'envoyer une requête à un autre agent (y compris à lui-même) et d'attendre la réponse de cet agent, soit de manière synchrone, soit de façon asynchrone à l'aide de Promise. Le second patron de communication est le patron Publisher/Subscriber qui permet à plusieurs agents de s'abonner à un type d'information produit par un agent. Lorsque l'agent déclenche l'événement ou produit la donnée attendue, tous les agents abonnés à ce type de données sont notifiés et reçoivent l'information. Cette

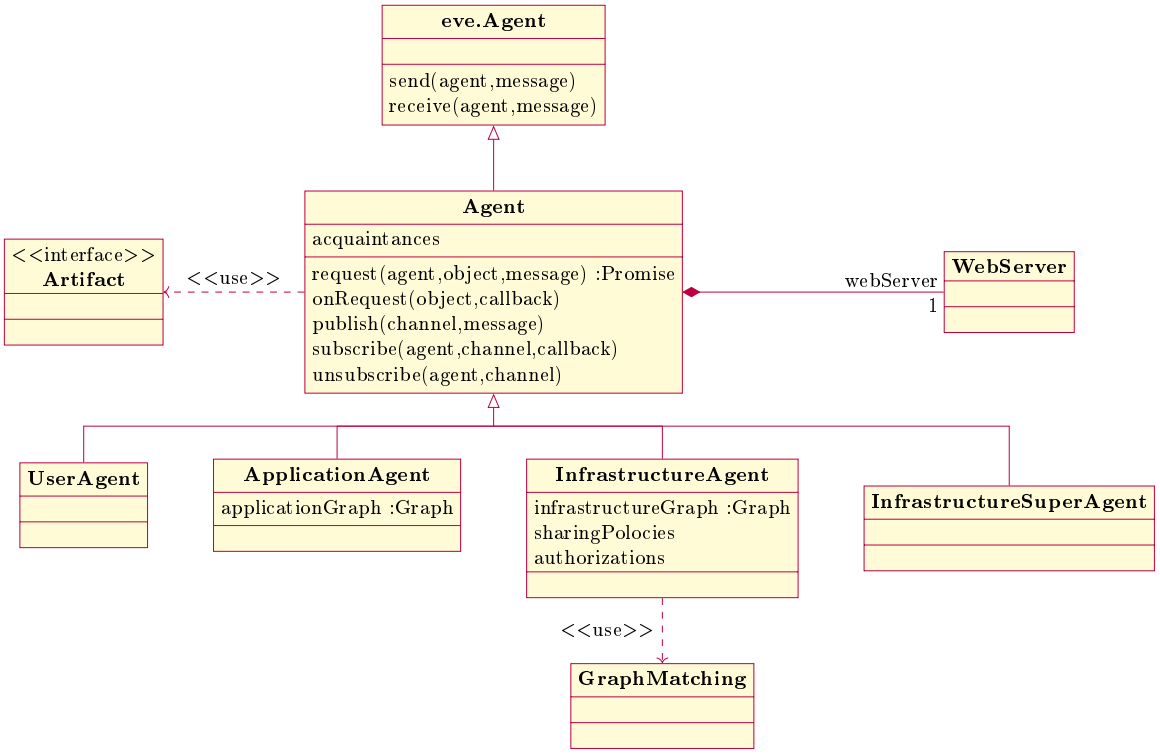


FIGURE 7.4 – Diagramme de classes UML de la partie agents

classe Agent embarque un serveur web représenté par la classe WebServeur qui met à disposition de l'utilisateur des interfaces permettant de dialoguer avec l'agent. Dans l'état actuel du développement, ces interfaces sont surtout utilisées pour déboguer nos agents. Néanmoins, dans une version future de l'intergiciel, ces interfaces permettront, moyennant un moyen d'identification adéquat, de gérer l'infrastructure et les applications déployées. Chaque agent possède la capacité d'utiliser des artifacts en utilisant l'interface Artifact mise à disposition. Ici aussi, l'état d'avancement actuel offre une interface très basique. Dans la suite des travaux une attention particulière sera apportée à l'étude de ces différents artifacts de déploiement. Enfin, les classes UserAgent,

ApplicationAgent, InfrastructureAgent et InfrastructureSuperAgent héritent des capacités de communication de la classe Agent et implémentent les comportements et protocoles de communication qui leur sont propres, selon les spécifications établies au Chapitre 6. Chaque agent s'exécute dans un processus NodeJS indépendant et peut donc fonctionner sur n'importe quelle machine supportant l'exécution de NodeJS.

7.3 Evaluation de l'algorithme

Cette section décrit les expérimentations effectuées avec l'intergiciel et en particulier sur la partie algorithmique, mettant en œuvre les interactions entre les agents ainsi que l'exécution de l'algorithme de graph-matching distribué.

7.3.1 Procédure de test

L'algorithme de graph-matching, dans sa version centralisée avec un unique agent infrastructure ou dans sa version distribuée faisant intervenir plusieurs agents infrastructure, a été évalué en faisant varier les différents paramètres caractérisant les graphes d'application et d'infrastructure. Chacun de ces graphes a donc été généré de manière aléatoire, en fonction de paramètres tels que :

— La taille du graphe d'application
— La taille du graphe d'infrastructure
— Le nombre moyen d'arcs par nœud
— Le nombre moyen de propriétés par nœud
— Le nombre de nœuds infrastructure par agent
— Le nombre d'agents infrastructure

Ceci permet d'étudier le comportement de l'algorithme indépendamment des contraintes liées au domaine d'application (ici les maisons intelligentes). Une fois son comportement général établi en fonction des différents paramètres, une étude des graphes d'infrastructure et d'application du domaine permettra de déterminer la plage des paramètres utilisée et d'en déduire les performances de l'algorithme dans ce cas.

Dans les sous-sections suivantes, chaque configuration de l'algorithme a été testée en générant cent graphes d'applications différents se projetant sur un graphe d'infrastructure aléatoirement généré. Chaque point du graphe correspond donc au temps médian que met l'algorithme pour trouver une projection d'un graphe d'application sur le graphe d'infrastructure dans ces cent cas.

7.3.2 Algorithme centralisé

Les premières expérimentations se sont concentrées sur le cas de l'algorithme centralisé. Nous avons vu au Chapitre 5 que l'algorithme décentralisé était basé sur sa version centralisée. Ainsi, l'intergiciel a donc pu être utilisé avec un unique agent infrastructure gérant la totalité de l'infrastructure matérielle. L'algorithme distribué se comporte donc comme sa version centralisée dans ce cas précis.

Les degrés de liberté pour évaluer la version centralisée de l'algorithme sont :
— La taille des graphes d'application et d'infrastructure
— Le nombre moyen d'arcs par nœud
— Le nombre moyen de propriétés par nœud

7.3.2.1 Taille des graphes

L'évolution des performances en fonction de la taille des graphes d'infrastructure et d'application a été testé ici. Les graphes ont été générés avec une moyenne de 5 arcs ainsi que 3 propriétés par nœud. La taille du graphe d'application varie de 10 à 80 nœuds et celle du graphe d'infrastructure de 100 à 10000.

La Figure 7.5 montre l'évolution du temps de calcul de la première projection en fonction de la taille du graphe d'infrastructure. Cette évolution semble être linéaire avec la taille du graphe d'infrastructure. Une régression linéaire indique une pente moyenne de $4.2e^{-3}$ quelle que soit la taille du graphe d'application.

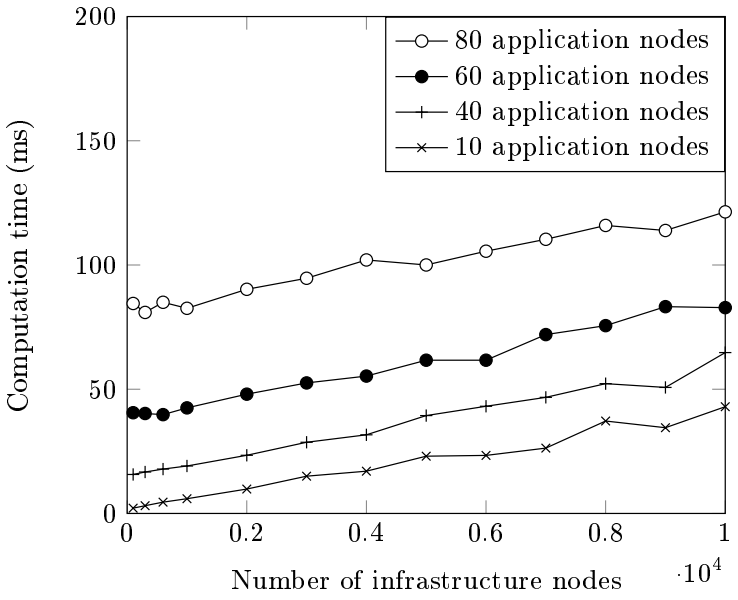


FIGURE 7.5 – Évolution du temps d'exécution de l'algorithme en fonction de la taille du graphe d'infrastructure pour différentes tailles du graphe d'application

La Figure 7.6 montre l'évolution du temps de calcul de la première projection en fonction de la taille du graphe d'application. Les données sont issues des mêmes expérimentations qui ont servi à tracer le graphique de la Figure 7.5, mais affichées différemment. Ce graphique montre maintenant une évolution qui semble être polynomiale avec la taille du graphe d'application, dans les régions étudiées.

L'algorithme offre de bonnes performances vis-à-vis de l'augmentation de la taille de l'infrastructure. Néanmoins, la vitesse de calcul d'une projection est polynomiale avec la taille des applications pour la plage de valeurs testée ici. Les performances de l'algorithme sont donc réduites avec l'augmentation de la complexité des applications

à déployer.

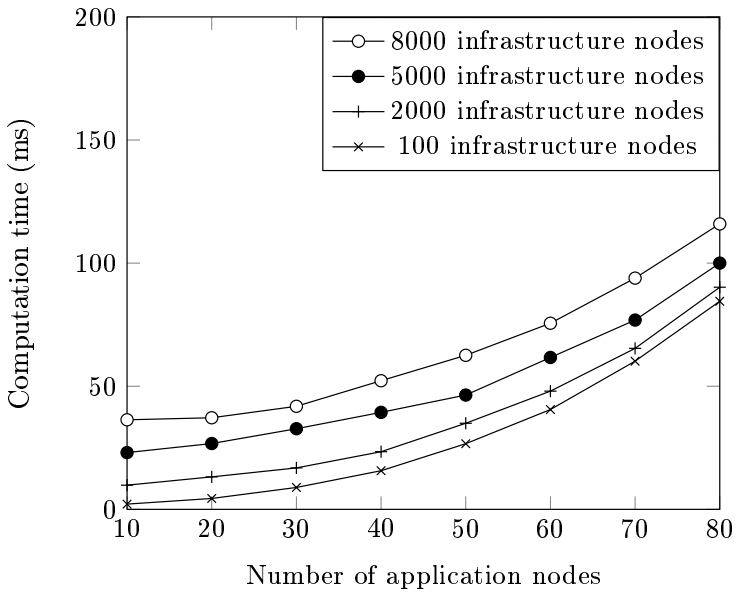


FIGURE 7.6 – Évolution du temps d'exécution de l'algorithme en fonction de la taille du graphe d'application pour différentes tailles du graphe d'infrastructure

7.3.2.2 Nombre d'arcs

Le critère principal testé ici est le nombre d'arcs moyen entre les nœuds des graphes d'application et d'infrastructure. Ce nombre d'arcs varie de 2 à 60 simultanément pour le graphe d'application et d'infrastructure. La taille du graphe d'infrastructure est variable de 200 à 5000 nœuds et celle des graphes d'application est fixe avec 80 nœuds. Les graphes ont été générés avec en moyenne 3 propriétés par nœud.

La Figure 7.7 représente l'évolution du temps de calcul en fonction du nombre moyen d'arcs par nœud pour plusieurs tailles du graphe d'infrastructure. Cette évolution est linéraire avec l'augmentation du nombre d'arcs par nœud avec une pente moyenne de 0.95 quelle que soit la taille du graphe d'application.

7.3.2.3 Nombre de propriétés

Enfin, le dernier critère à tester pour la version centralisée de l'algorithme est celui du nombre moyen de propriétés par nœud. Ici, le nombre de propriétés des graphes d'application et d'infrastructure varie de 2 à 9. Le nombre moyen d'arcs entre les nœuds est de 5 et la taille du graphe d'application est fixé à 50 nœuds. La taille du graphe d'infrastructure varie de 200 à 5000 nœuds.

Figure 7.8 montre l'évolution du temps de calcul d'une projection en fonction du nombre moyen de propriétés par nœud pour différentes tailles du graphe d'application.

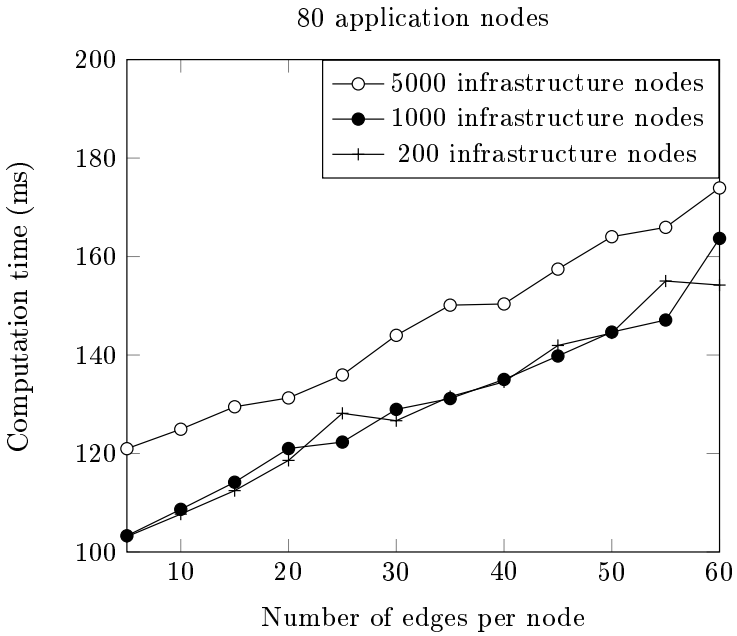


FIGURE 7.7 – Évolution du temps d'exécution de l'algorithme en fonction du nombre d'arcs par nœud pour différentes tailles du graphe d'infrastructure

Cette évolution est linéaire avec l'augmentation du nombre de propriétés par nœud, mais le coefficient de linéarité augmente avec la taille du graphe d'infrastructure.

7.3.3 Algorithme décentralisé

Les expérimentations suivantes concernent l'algorithme dans sa version distribuée, c'est-à-dire avec plusieurs agents coopérant entre-eux. Ici, il est possible de jouer sur le nombre d'agents infrastructure en fonction de la taille des graphes. Dans les résultats qui suivent, le nombre moyen d'arcs par nœud est fixé à 5 et le nombre de propriétés à 3 par nœud.

7.3.3.1 Nombre d'agents et taille de l'infrastructure

Le premier jeu de tests cherche à déterminer les performances de l'algorithme en fonction du nombre d'agents infrastructure pour plusieurs tailles du graphe d'infrastructure. Le nombre d'agents varie de 1 (algorithme centralisé) à 50 pour des tailles de l'infrastructure allant de 200 à 1000. La taille du graphe d'application est fixée à 50 nœuds.

Figure 7.9 représente l'évolution du temps de calcul d'une première projection en fonction du nombre d'agents pour des tailles d'infrastructure variables. Nous pouvons noter que le temps de calcul augmente avec le nombre d'agents dans le système. Il est par contre surprenant de noter que les temps de calcul les plus importants correspondent aux cas où l'infrastructure comporte très peu de nœuds. Dans ces cas-là les

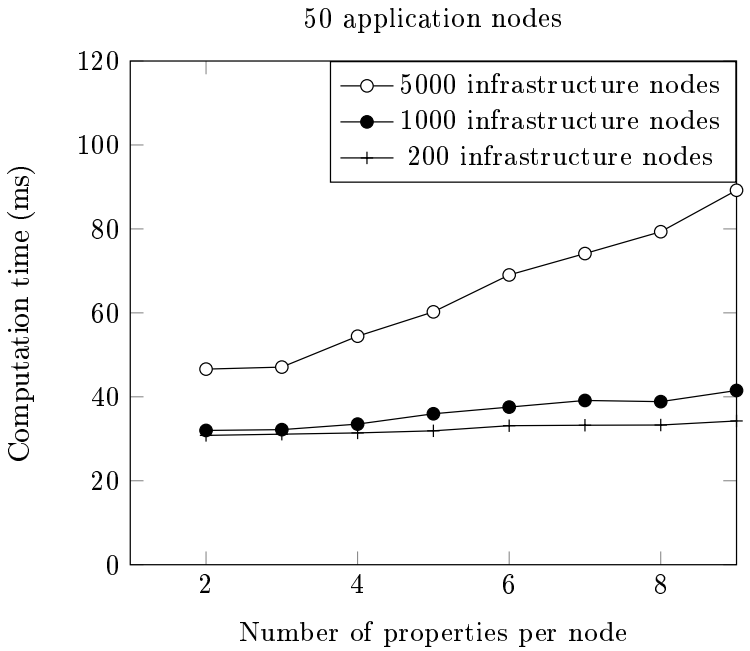


FIGURE 7.8 – Évolution du temps d'exécution de l'algorithme en fonction du nombre d'arcs par nœud pour différentes tailles du graphe d'infrastructure

agents infrastructure prennent en charge très peu de nœuds (4 nœuds pour le cas d'un graphe d'infrastructure de 200 nœuds répartis entre 50 agents). Le graphe d'application possédant toujours 50 nœuds, les agents infrastructure passent leur temps à coopérer plutôt qu'à calculer la solution. Le temps de communication augmente beaucoup vis-à-vis du temps de calcul.

Pour mieux illustrer ce phénomène, nous avons tracé à la Figure 7.10 l'évolution du temps de calcul en fonction de la taille du graphe d'infrastructure pour différents nombres d'agents. Pour un nombre d'agents infrastructure important, le temps de calcul est inversement proportionnel au nombre de nœuds infrastructure pris en charge par chaque agent. Il est possible d'en conclure qu'une agentification très fine de l'infrastructure n'est pas souhaitable. Même si en théorie, avoir un agent par entité matérielle peut sembler élégant, les agents passeraient trop de temps à interagir entre eux. Il vaut mieux privilégier des agents gérant un ensemble d'entités matérielles localisées dans l'espace (un agent par pièce d'un logement ou un agent par logement par exemple). Ainsi, les agents, privilégiant un déploiement local, interagissent uniquement avec les agents de leur voisinage.

7.3.3.2 Nombre d'agents et taille des applications

Le second jeu de tests cherche à déterminer les performances de l'algorithme en fonction du nombre d'agents infrastructure pour différentes tailles d'applications. Le nombre d'agents reste compris entre 1 et 50, la taille des applications varie de 10 à 60 et la taille de l'infrastructure est fixée à 500 nœuds.

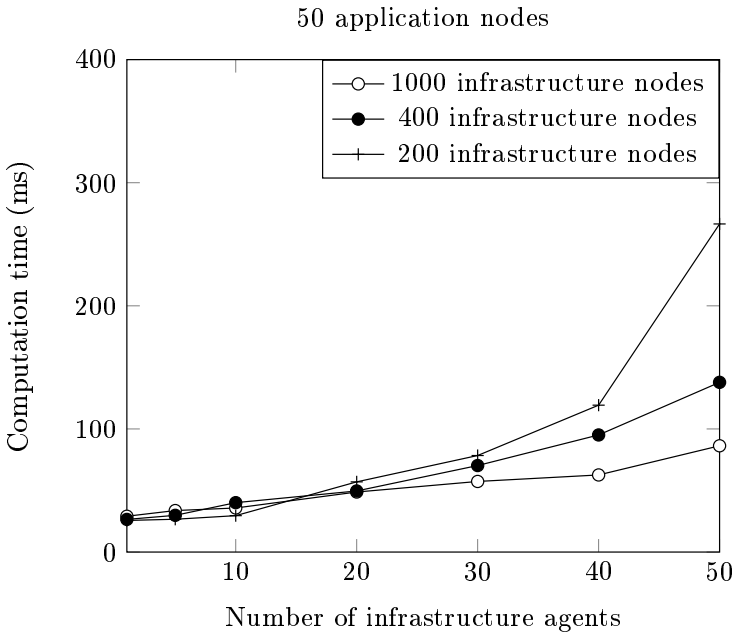


FIGURE 7.9 – Évolution du temps d'exécution de l'algorithme en fonction du nombre d'agents impliqués pour différentes tailles de graphes d'infrastructure

Figure 7.11 montre l'évolution du temps de calcul en fonction du nombre d'agents pour des tailles d'applications différentes. Sans surprise, le temps de calcul accroît de manière plutôt linéaire en fonction du nombre d'agents.

La Figure 7.12 réutilise les mêmes données pour tracer l'évolution du temps de calcul cette fois-ci en fonction de la taille des applications pour différents nombres d'agents. Dans le cas d'un système à un agent, l'on retrouve une évolution polynomiale. Dans les cas où il y a plusieurs agents, le type d'évolution est moins triviale.

7.3.4 Corrélation avec le domaine d'application

L'étude des graphes de l'infrastructure et des applications utilisées pour décrire un environnement intelligent de type maison nous a permis d'extraire leurs propriétés.

Ainsi, la taille des graphes d'application reste en général relativement faible (une vingtaine de nœuds environ). Le nombre moyen de relations entrantes ou sortantes par nœud est compris entre 2 et 4, mis à part pour les nœuds de type Networks qui peuvent posséder beaucoup plus de relations. Le nombre de propriétés par nœud varie de 1 à 4 avec la version de l'ontologie présentée au Chapitre 3. La taille du graphe de l'infrastructure, quant à elle, peut augmenter de manière assez significative. Pour un logement, il est possible d'atteindre une centaine de nœuds qui peuvent être répartis entre plusieurs agents.

Étant donné que l'algorithme évolue de manière linéaire avec la taille du graphe d'infrastructure, mais exponentiellement avec la taille du graphe d'application, le fait d'avoir des graphes d'application de petite taille et des graphes d'infrastructure de taille

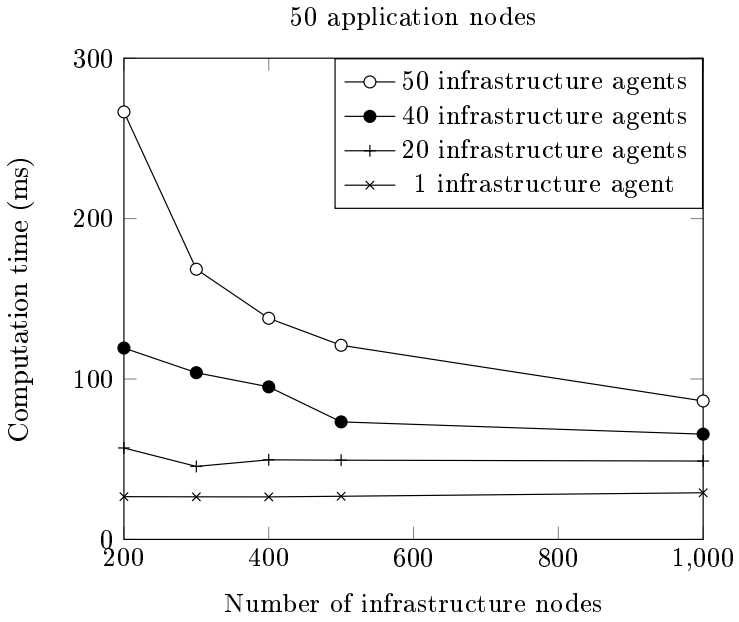


FIGURE 7.10 – Évolution du temps d'exécution de l'algorithme en fonction de la taille du graphe de l'infrastructure prise en charge par différents nombres d'agents

plus importante est ici un atout permettant d'utiliser l'algorithme de manière performante. De plus, le nombre moyen réduit de relations entre les nœuds et de propriétés par nœud est un second argument en faveur de l'utilisation de cet algorithme dans des environnements réels. Enfin, le type d'agentification par espace (pièce, logement) permet de privilégier des coopérations à un niveau local, empêchant ainsi une perte de temps trop importante due à la communication entre les agents.

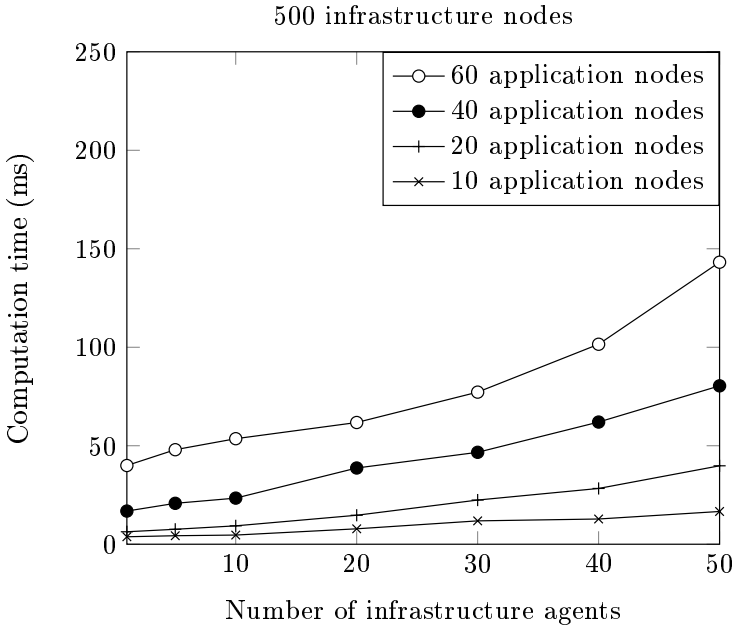


FIGURE 7.11 – Évolution du temps d'exécution de l'algorithme en fonction du nombre d'agents infrastructure pour différentes tailles du graphe d'application

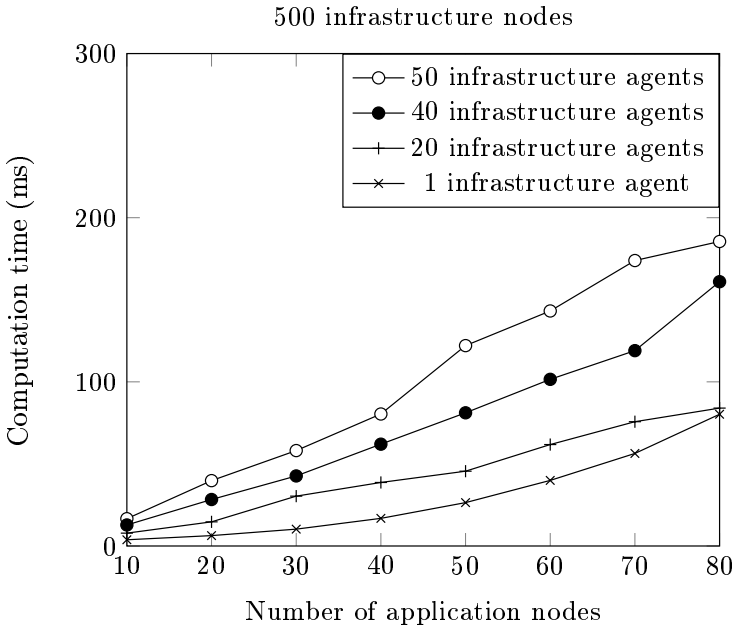


FIGURE 7.12 – Évolution du temps d'exécution de l'algorithme en fonction de la taille du graphe d'application pour différents nombres d'agents infrastructure

7.4 Conclusion

Dans ce chapitre, nous avons présenté le cadre expérimental dans lequel les propositions de ce travail de thèse sont en cours d'évaluation. Nous avons introduit l'intergiciel de déploiement d'applications pour des environnements ambiants et nous avons expliqué son fonctionnement ainsi que les choix technologiques associées. La fin de ce chapitre est consacrée aux expérimentations elles-mêmes. Nous avons évalué les performances de l'algorithme centralisé et distribué dans le cadre général. Nous avons montré qu'appliqué aux cas des maisons intelligentes, ces résultats étaient très encourageants et que la solution pouvait passer à l'échelle afin de gérer des ensembles de logements regroupés au sein de plusieurs immeubles.

Conclusion

Les progrès technologiques et la miniaturisation continue des périphériques électroniques nous permettent de concevoir d'un côté une infrastructure énergétique, réseau et matérielle capables de supporter des applications ubiquitaires et d'un autre côté, des applications intelligentes et context-aware qui sont développées pour aider l'utilisateur de manière la moins intrusive possible. L'Internet des Objets et l'Intelligence Ambiante sont des domaines qui sont les catalyseurs de la prochaine révolution des technologies de l'information. Le premier se focalise sur le niveau des infrastructures alors que la seconde travaille au niveau des applications. Néanmoins, nous avons montré qu'il reste encore un fossé entre cette infrastructure matérielle et les applications intelligentes : peu de choses existent actuellement pour permettre de raisonner sur l'hétérogénéité de ces systèmes ambiants ou sur le déploiement d'applications sur l'infrastructure existante.

La problématique de cette thèse s'inscrit dans la recherche d'une solution permettant le déploiement et la configuration automatique d'applications distribuées dans des environnements intelligents. Nous avons situé nos travaux à l'intersection des domaines de l'Intelligence Ambiante ainsi que de l'Internet des Objets. Nous nous sommes intéressés en particulier à quatre sous-problèmes :

— L'hétérogénéité des entités matérielles et logicielles
— La dynamicité des environnements ambiants
— Le passage à l'échelle
— La confidentialité des informations relatives à l'utilisateur

Ces quatre sous-problèmes ont guidé tout au long de ce travail l'élaboration de la solution proposée dans ce mémoire. L'hétérogénérité fait référence à la diversité des technologies et des protocoles de communication entre les entités matérielles et logicielles, mais aussi au nombre important de contraintes à prendre en compte (mobilité, énergétique, communication, etc.). La dynamicité des environnements implique que ceux-ci changent constamment et que la solution doit s'adapter dynamiquement à ces changements. Le passage à l'échelle implique une conception décentralisée et distribuée de la solution permettant à tout moment l'ajout d'entités (entités matérielles, logements, bâtiments, etc.). Enfin, la confidentialité implique qu'il faille fournir des mécanismes garantissant que les données de l'utilisateur puissent rester privées. Nous avons choisi de nous intéresser en particulier aux informations concernant la structure et les propriétés de l'infrastructure matérielle de l'utilisateur.

Les travaux étudiés dans l'état de l'art nous ont permis d'une part, de comprendre

la manière dont les travaux actuels gèrent les environnements ambiants afin de couvrir une ou plusieurs des caractéristiques énoncées ci-dessus. D'autre part, nous avons aussi étudié les différentes approches pour le déploiement et la configuration d'applications dans les domaines des services, des grilles de calculs ainsi que du Cloud Computing. Les solutions proposées ne sont certes pas directement applicables au problème du déploiement d'applications dans des environnements ambiants, néanmoins un certain nombre de mécanismes et d'architectures ont pu inspirer l'élaboration de la solution proposée dans ce mémoire.

Contributions

Notre proposition est constituée d'une approche pour le déploiement d'applications au sein d'un environnement ambiant aboutissant à un intergiciel agent. Les principales contributions sont la proposition :

— d'un métamodèle permettant de décrire l'environnement ambiant en terme d'infrastructure matérielle disponible et d'applications déployables. L'infrastructure matérielle est décrite sous la forme d'un graphe orienté dans lequel les nœuds correspondent aux entités de l'infrastructure (matérielles, logicielles et virtuelles) auxquelles il est possible d'accrocher un ensemble de propriétés caractérisant l'entité. Les entités sont reliées entre elles par des relations apportant une information sémantique sur le type de lien. Les applications déployables sont décrites sous la forme d'un graphe que l'on peut découper en deux parties. La première partie décompose l'application en différentes fonctionnalités et nous informe sur les interactions entre ces fonctionnalités. La seconde partie quant à elle correspond au patron de l'application : le graphe correspondant aux besoins de l'application en terme de matériel pour pouvoir être déployé correctement.

— d'une formalisation mathématique à base d'homomorphismes de graphes qui permet de raisonner sur les descriptions de l'environnement afin de trouver dans le graphe de l'infrastructure les entités qui correspondent aux besoins du patron de l'application.

— d'un algorithme de graph matching distribué permettant de trouver l'un de ces homomorphismes tout en garantissant la confidentialité des informations contenues dans le graphe de l'infrastructure. Ce dernier est distribué entre différentes entités qui ne se partagent pas les informations entre elles. Pour coopérer, ces entités s'échangent des projections partielles qui indiquent uniquement les parties du patron de l'application qui ont été assignées et celles qui ne le sont pas encore.

— d'une architecture multi-agents ainsi que d'un modèle d'agent permettant d'une part une séparation claire entre les utilisateurs, les applications et l'infrastructure et d'autre part l'intégration de mécanismes supplémentaires à base de politiques de partage permettant de renforcer la confidentialité des ressources des utilisateurs.

Toutes ces contributions ont été regroupées au sein d'un intergiciel permettant la description de l'infrastructure et des applications, l'établissement de plans de déploiement de ces applications sur l'infrastructure et l'exécution de ces plans de déploiement à l'aide d'un ensemble d'artifacts.

Nous avons vu que cette solution prenait en compte l'hétérogénéité des entités de l'environnement, autorisant ainsi à tirer parti au mieux des caractéristiques intrinsèques à ces entités, sans vouloir passer par une couche interopérable qui masque ces spécificités. Nous nous sommes aussi attardés sur la confidentialité des informations relatives à l'infrastructure matérielle de l'utilisateur. Nous avons pour cela proposé un algorithme distribué qui permet de trouver une projection d'un patron d'une application sur l'infrastructure sans partager les informations de cette infrastructure de manière globale, renforcé par des politiques de partage permettant à l'utilisateur d'avoir le contrôle sur l'utilisation de ses ressources. Enfin, la dynamicité et le passage à l'échelle ont été pris en compte grâce au paradigme agent qui nous a permis d'élaborer une solution distribuée et décentralisée dans laquelle les agents et les artifacts raisonnent et communiquent localement pour élaborer les plans de déploiement et déployer effectivement une application. L'aspect dynamique a été pris en compte grâce aux artifacts de monitoring permettant de remonter des informations contextuelles sur l'environnement et ainsi planifier un redéploiement des applications devenues inconsistantes.

Perspectives

La proposition d'un intergiciel agent pour le déploiement d'applications dans des environnements ambiants, sa mise en œuvre ainsi que les premiers résultats d'expérimentations ont donné lieu à un certain nombre de réflexions. Cette sous-section a pour objectif de critiquer mes travaux actuels et de proposer quelques perspectives pour son amélioration.

Le métamodèle proposé a été conçu pour décrire des environnements ambiants de type maison. Une première perspective serait d'enrichir ce métamodèle pour prendre en compte les caractéristiques d'autres environnements tels que des bâtiments ou des villes par exemple. Cet enrichissement permettra de mieux structurer l'ontologie et d'extraire un certain nombre de concepts communs à tous types d'environnement comme par exemple, la notion de fonctionnalité. L'aspect contextuel a été traité à l'aide de relations permettant d'indiquer la présence ou non d'entités ou d'utilisateurs dans un endroit géographique. Ici aussi, l'ontologie peut être enrichie afin de prendre en compte des informations contextuelles plus variées servant à raisonner sur le déploiement d'applications. Bon nombre de travaux de mes prédécesseurs et notamment ceux de Olaru [Olaru 11], se focalisent uniquement sur l'aspect contextuel qui constitue un domaine de recherche à part entière.

Dans ce travail, je me suis concentré sur l'utilisation du graphe de l'infrastructure et des patrons des applications pour raisonner sur le déploiement et la configuration de ces applications. J'ai néanmoins proposé au Chapitre 3 une description plus complète des applications à l'aide de fonctionnalités interagissant entre elles. Même si cette partie descriptive n'a pas véritablement servi dans la suite de ces travaux, elle permet d'ouvrir la porte à la proposition de raisonnement concernant les types d'interactions et les données transitant entre ces fonctionnalités. Moyennant un enrichissement de l'ontologie, il est possible de décrire plus précisément le type de données transitant entre les différentes fonctionnalités de l'application et ainsi permettre à l'utilisateur de définir quelles sont les données qu'il autorise à sortir de son logement et quelles sont

celles qu'il veut absolument traiter localement.

Au Chapitre 4, je définis des opérateurs permettant de combiner (Section 4.1.1.2) et comparer (Section 4.1.1.3) différents types de propriétés caractérisant les entités de l'environnement. Ces opérateurs ont été réutilisés au Chapitre 5 dans la définition de l'itérateur trouvant les nœuds compatibles (Section 5.1.2). Néanmoins, je me suis assez peu intéressé à l'interaction entre les propriétés des nœuds, c'est-à-dire aux contraintes de ces propriétés. J'ai considéré qu'on laisse à la charge du programmeur la définition de ces interactions. Il est pourtant possible de s'intéresser à ces mécanismes et de proposer un certain nombre d'opérateurs et de contraintes génériques possédant diverses propriétés.

Les problèmes temporels d'allocation de ressources ont été totalement évacués de nos recherches. Une grande quantité de travaux traitant de ce problème existe déjà. Sa prise en considération aurait modifié de manière conséquente l'orientation de nos recherches. Nous avons voulu répondre au problème du déploiement d'applications dans des environnements ambiants et non à celui de l'allocation de ressources. Néanmoins, une perspective à moyen ou à long terme serait d'étendre l'ontologie afin de rajouter des informations contextuelles sur l'utilisation ou non de ressources d'un point de vue temporel. Couplé par exemple à des mécanismes de priorités entre les applications, cela pourrait constituer une approche originale au problème d'allocation de ressources entre applications.

L'apport principal de mon travail de thèse est la proposition de mécanismes permettant de déterminer les entités de l'infrastructure à utiliser pour déployer des applications. Néanmoins, le déploiement en tant que tel n'a pas été approfondi. J'ai proposé au Chapitre 6 des artifacts de déploiement permettant d'interagir avec l'environnement pour déployer des logiciels et configurer du matériel. Comme nous l'avons vu dans l'état de l'art, le déploiement d'applications est un domaine très fructueux depuis les années 1990. Il est donc possible de s'inspirer et même de réutiliser ces travaux pour concevoir les artifacts qui serviront au déploiement effectif.

Enfin, le système multi-agents constituant le cœur de l'intergiciel se trouve être en dehors du système. Une perspective serait de considérer le SMA comme étant une application à part entière totalement déployable, ce qui permettrait de bootstrapper le système à l'aide d'un seul agent de base.

Bien évidemment, le développement de l'intergiciel ainsi que la mise en œuvre de la solution dans des environnements réels constituent un travail important à court terme. L'implémentation des interfaces de monitoring et des outils de développement est une priorité.

Liste des publications

F. Piette, C. Dinont, A. El Fallah-Seghrouchni, P. Taillibert. *Deployment and configuration of applications for ambient systems.* Procedia Computer Science. In Proceedings of the 6th International Conference on Ambient Systems, Networks and Technologies (ANT), United-Kingdom, 2015.

F. Piette, C. Caval, C. Dinont, A. El Fallah-Seghrouchni, P. Taillibert. *A Multi-Agent System for the Deployment of Applications in Ambient Systems.* Short paper - International Conference on Autonomous Agents and Multi-Agent Systems (AAMAS), Singapore, 2016.

F. Piette, C. Caval, C. Dinont, A. El Fallah-Seghrouchni, P. Taillibert. *A Multi-Agent Solution for the Deployment of Distributed Applications in Ambient Systems.* Revised Selected Papers of the 4th International Workshop on Engineering Multi-Agent Systems (EMAS), Singapore, 2016.

F. Piette, C. Caval, C. Dinont, A. El Fallah-Seghrouchni, P. Taillibert. *A Multi-Agent Approach for the Deployment of Distributed Applications in Smart Environments.* Selected for publication in a Journal - Special issue of the International Conference on Intelligent Distributed Computing (IDC), France, 2016.

F. Piette, A. El Fallah-Seghrouchni, P. Taillibert. *Intelligent Agents for Preserving Resource Privacy when Deploying Ambient Intelligence Applications.* In Proceedings of International Conference on Agents (ICA), Japan, 2016.

F. Piette, C. Caval, C. Dinont, A. El Fallah-Seghrouchni, P. Taillibert. *Deployment of ambient applications in smart environments.* Workshop on Smart and Sustainable City (WSSC), France, 2016.

F. Piette, A. El Fallah-Seghrouchni, P. Taillibert, C. Caval, C. Dinont. *A Multi-Agent Middleware For Deployment Of Ambient Applications.* Book chapter in Enablers for Smart Cities, ISTE, 2016.

Bibliographie

[Aïmeur 06] E. Aïmeur, G. Brassard, J. M. Fernandez & F. S. Mani Onana. *Privacy-preserving Demographic Filtering*. In Proceedings of the 2006 ACM Symposium on Applied Computing, SAC '06, pages 872–878, New York, NY, USA, 2006. ACM. 29

[Alberola 10] JuanM. Alberola, JoseM. Such, Ana Garcia-Fornes, Agustín Espinosa & Vicent Botti. *A performance evaluation of three multiagent platforms*. Artificial Intelligence Review, vol. 34, no. 2, pages 145–176, 2010. 29

[Armbrust 09] Michael Armbrust, Armando Fox, Rean Griffith, Anthony D Joseph, Randy H Katz, Andrew Konwinski, Gunho Lee, David A Patterson, Ariel Rabkin, Ion Stoica*et al.* *Above the clouds : A berkeley view of cloud computing*. 2009. 39

[Arnold 07] William Arnold, Tamar Eilam, Michael H. Kalantar, Alexander V. Konstantinou & Alexander Totok. *Pattern Based SOA Deployment*. In Bernd J. Kramer, Kwei-Jay Lin & Priya Narasimhan, editeurs, ICSOC, volume 4749 of *Lecture Notes in Computer Science*, pages 1–12. Springer, 2007. 33

[Arnold 08] William Arnold, Tamar Eilam, Michael Kalantar, AlexanderV. Konstantinou & AlexanderA. Totok. *Automatic Realization of SOA Deployment Patterns in Distributed Environments*. In Athman Bouguettaya, Ingolf Krueger & Tiziana Margaria, editeurs, Service-Oriented Computing – ICSOC 2008, volume 5364 of *Lecture Notes in Computer Science*, pages 162–179. Springer Berlin Heidelberg, 2008. 34

[Babai 15] László Babai. *Graph Isomorphism in Quasipolynomial Time*. CoRR, vol. abs/1512.03547, 2015. 51

[Babu 14] AB Karthick Anand Babu & R Sivakumar. *Development of ontology based middleware for context awareness in ambient*

intelligence. In Hybrid Intelligent Systems (HIS), 2014 14th International Conference on, pages 219–224. IEEE, 2014. 26

[Babu 15] AB Karthick Anand Babu & R Sivakumar. *Development of Type 2 Fuzzy Rough Ontology-based Middleware for Context Processing in Ambient Smart Environment*. In Intelligent Computing and Applications, pages 137–143. Springer, 2015. 26

[Baduel 06] Laurent Baduel, Françoise Baude, Denis Caromel, Arnaud Contes, Fabrice Huet, Matthieu Morel & Romain Quilici. *Programming, composing, deploying for the grid*. In Grid Computing : Software Environments and Tools, pages 205–229. Springer, 2006. 37

[Barth 06] A. Barth, A. Datta, J.C. Mitchell & H. Nissenbaum. *Privacy and contextual integrity : framework and applications*. In Security and Privacy, 2006 IEEE Symposium on, pages 15 pp.–198, May 2006. 28

[Bergmayr 15] Alexander Bergmayr, Alessandro Rossini, Nicolas Ferry, Geir Horn, Leire Orue-Echevarria, Arnor Solberg & Manuel Wimmer. *The Evolution of CloudML and its Manifestations*. In Proceedings of the 3rd International Workshop on Model-Driven Engineering on and for the Cloud (CloudMDE), pages 1–6, 2015. 42

[Berners-Lee 90] Tim Berners-Lee & Robert Cailliau. *WorldWideWeb : Proposal for a HyperText project*. Retrieved on February, vol. 26, page 2008, 1990. 54

[Berners-Lee 01] Tim Berners-Lee, James Hendler, Ora Lassila*et al.* *The semantic web*. Scientific american, vol. 284, no. 5, pages 28–37, 2001. 49, 54

[Bessiere 94] Christian Bessiere. *Arc-consistency and arc-consistency again*. Artificial intelligence, vol. 65, no. 1, pages 179–190, 1994. 53

[Böhlen 10] Marc Böhlen & Hans Frei. *Ambient Intelligence in the City overview and new perspectives*. In Handbook of Ambient Intelligence and Smart Environments, pages 911–938. Springer, 2010. 20

[Bossardt 03] Matthias Bossardt, Andreas Mühlemann, Reto Zürcher & Bernhard Plattner. *Pattern based service deployment for active networks*. In Proc. Intern. Workshop Active Network Technol. and Appl. Citeseer, 2003. 33

[Bouchenak 06] Sara Bouchenak, Noel De Palma, Daniel Hagimont & Christophe Taton. *Autonomic management of clustered applications*. In 2006 IEEE International Conference on Cluster Computing, pages 1–11. IEEE, 2006. 38

[Brachman 79] Ronald Jay Brachman. *On the epistemological status of semantic networks*. Associative networks : Representation and use of knowledge by computers, pages 3–50, 1979. 51

[Bramley 00] Randall Bramley, Kenneth Chiu, Shridhar Diwan, Dennis Gannon, Madhusudhan Govindaraju, Nirmal Mukhi, Benjamin Temko & Madhuri Yechuri. *A component based services architecture for building distributed applications*. In High-Performance Distributed Computing, 2000. Proceedings. The Ninth International Symposium on, pages 51–59. IEEE, 2000. 37

[Brandtzæg 12] Eirik Brandtzæg, Sébastien Mosser & Parastoo Mohagheghi. *Towards CloudML, a model-based approach to provision resources in the clouds*. In 8th European Conference on Modelling Foundations and Applications (ECMFA), pages 18–27, 2012. 41

[Breivold 07] Hongyu Pei Breivold & Magnus Larsson. *Component-based and service-oriented software engineering : Key concepts and principles*. In 33rd EUROMICRO Conference on Software Engineering and Advanced Applications (EUROMICRO 2007), pages 13–20. IEEE, 2007. 31

[Brickley 04] Dan Brickley & Ramanathan V Guha. *RDF vocabulary description language 1.0 : RDF schema*. 2004. 56

[Cabri 05] Giacomo Cabri, Luca Ferrari, Letizia Leonardi & Franco Zambonelli. *The LAICA project : Supporting ambient intelligence via agents and ad-hoc middleware*. In 14th IEEE International Workshops on Enabling Technologies : Infrastructure for Collaborative Enterprise (WETICE'05), pages 39–44. IEEE, 2005. 24

[Caire 03] F Bellifemine G Caire, A Poggi & G Rimassa. *JADE : A White Paper*, 2003. 23

[Cao 09] Qi Cao, Zhi-Bo Wei & Wen-Mao Gong. *An optimized algorithm for task scheduling based on activity based costing in cloud computing*. In 2009 3rd International Conference on Bioinformatics and Biomedical Engineering, pages 1–3. IEEE, 2009. 41

[Caron 04] Eddy Caron, Pushpinder-Kaur Chouhan & Arnaud Legrand. *Automatic deployment for hierarchical network enabled servers*. In Parallel and Distributed Processing Symposium, 2004. Proceedings. 18th International, page 109. IEEE, 2004. 37

[Caron 06] Eddy Caron, Pushpinder Kaur Chouhan & Holly Dail. *GoDIET : A deployment tool for distributed middleware on grid'5000*. PhD thesis, INRIA, 2006. 37

[Carzaniga 98] Antonio Carzaniga, Alfonso Fuggetta, Richard S Hall, Dennis Heimbigner, Andre Van Der Hoek & Alexander L Wolf. *A characterization framework for software deployment technologies*. Rapport technique, DTIC Document, 1998. 30, 32

[Caval 14] Costin Caval, Amal El Fallah Seghrouchni & Patrick Taillibert. *Keeping a Clear Separation between Goals and Plans.* In Fabiano Dalpiaz, Jürgen Dix & M.Birna van Riemsdijk, editeurs, Engineering Multi-Agent Systems, volume 8758 of *Lecture Notes in Computer Science*, pages 15–39. Springer International Publishing, 2014. 126

[Ceccato 60] Silvio Ceccato & ENRICO MARETTI. *Linguistic analysis and programming for mechanical translation (mechanical translation and thought).* Rapport technique, DTIC Document, 1960. 49

[Change 12] IBM Tivoli Change. *Configuration Management Database (CCMDB), IBM Corporation*, 2012. 35

[Charif-Djebbar 06] Yasmine Charif-Djebbar & Nicolas Sabouret. *An agent interaction protocol for ambient intelligence.* In Intelligent Environments, 2006. IE 06. 2nd IET International Conference on, volume 1, pages 275–284. IET, 2006. 25

[Chein 08] Michel Chein & Marie-Laure Mugnier. Graph-based knowledge representation : Computational foundations of conceptual graphs. Springer, London, 2008. 45, 90

[Chen 04] Harry Chen, Timothy W. Finin, Anupam Joshi, Lalana Kagal, Filip Perich & Dipanjan Chakraborty 0001. *Intelligent Agents Meet the Semantic Web in Smart Spaces.* vol. 8, no. 6, pages 69–79, 2004. 11, 21, 27

[Cheong 06] Christopher Cheong & Michael Winikoff. Agent-oriented software engineering vi : 6th international workshop, aose 2005, utrecht, the netherlands, july 25, 2005. revised and invited papers, chapitre Hermes : Designing Goal-Oriented Agent Interactions, pages 16–27. Springer Berlin Heidelberg, Berlin, Heidelberg, 2006. 126

[Chopra 07] Samir Chopra & Laurence White. *Privacy and artificial agents, or, is google reading my email ?* In IJCAI, pages 1245–1250, 2007. 16, 29

[Cissée 07] Richard Cissée & Sahin Albayrak. *An Agent-based Approach for Privacy-preserving Recommender Systems.* In Proceedings of the 6th International Joint Conference on Autonomous Agents and Multiagent Systems, AAMAS '07, pages 182 :1–182 :8, New York, NY, USA, 2007. ACM. 29

[Conte 04] Donatello Conte, Pasquale Foggia, Carlo Sansone & Mario Vento. *Thirty Years Of Graph Matching In Pattern Recognition.* IJPRAI, vol. 18, no. 3, pages 265–298, 2004. 50, 85

[Coppola 07] Massimo Coppola, Nicola Tonellotto, Marco Danelutto, Corrado Zoccolo, Sebastien Lacour, Christian Perez & Thierry Priol. *Towards a common deployment model for grid systems.* In Integrated research in GRID computing, pages 15–30. Springer, 2007. 38

[Cordella 01] L. P. Cordella, P. Foggia, C. Sansone & M. Vento. *An improved algorithm for matching large graphs.* In In : 3rd IAPR-TC15 Workshop on Graph-based Representations in Pattern Recognition, Cuen, pages 149–159, 2001. 50

[Corneil 70] Derek G Corneil & Calvin C Gotlieb. *An efficient algorithm for graph isomorphism.* Journal of the ACM (JACM), vol. 17, no. 1, pages 51–64, 1970. 50

[Costantini 08] Stefania Costantini, Leonardo Mostarda, Arianna Tocchio & Panagiota Tsintza. *DALICA : Agent-Based Ambient Intelligence for Cultural-Heritage Scenarios.* vol. 23, no. 2, pages 34–41, 2008. 21

[Crépin 09] Ludivine Crépin, Yves Demazeau, Olivier Boissier & François Jacquenet. *Sensitive Data Transaction in Hippocratic Multi-Agent Systems.* In Engineering Societies in the Agents World IX, Lecture Notes in Computer Science, pages 85–101. Springer Berlin / Heidelberg, 2009. 27

[Curbera 05] Francisco Curbera, Donald Ferguson, Martin Nally & Marcia L Stockton. *Toward a programming model for service-oriented computing.* In International Conference on Service-Oriented Computing, pages 33–47. Springer, 2005. 31

[Danelutto 06] Marco Danelutto, Marco Vanneschi, Corrado Zoccolo, Nicola Tonellotto, Salvatore Orlando, Ranieri Baraglia, Tiziano Fagni, Domenico Laforenza & Alessandro Paccosi. *HPC application execution on grids.* In Future Generation Grids, pages 263–282. Springer, 2006. 37

[De Jong 89] Kenneth A De Jong & William M Spears. *Using Genetic Algorithms to Solve NP-Complete Problems.* In ICGA, pages 124–132, 1989. 51

[Dean 04] Jeffrey Dean & Sanjay Ghemawat. *MapReduce : Simplified Data Processing on Large Clusters.* 0018-9162/95 D OSDI IEEE, 2004. 14

[Deng 04] Gan Deng, Aniruddha Gokhale & Balachandran Natarajan. *Model-driven integration of federated event services in real-time component middleware.* In Proceedings of the 42nd annual Southeast regional conference, pages 353–356. ACM, 2004. 33

[Denis 03a] Alexandre Denis, Christian Pérez & Thierry Priol. *Achieving portable and efficient parallel CORBA objects.* Concurrency and Computation : Practice and Experience, vol. 15, no. 10, pages 891–909, 2003. 37

[Denis 03b] Alexandre Denis, Christian Pérez, Thierry Priol & André Ribes. *Padico : A component-based software infrastructure for grid computing.* In Parallel and Distributed Processing Symposium, 2003. Proceedings. International, pages 8–pp. IEEE, 2003. 37

[Donsez 07] Didier Donsez. *On-demand component deployment in the upnp device architecture.* In 4th IEEE Consumer Communications and Networking Conference (CCNC'07), Las Vegas, NV, USA, 2007. 31

[Donsez 10] Didier Donsez, Stéphane Chomat, Kiev Gama, Walter Rudametkin & Lionel Touseau. *A Service-Oriented Platform for iTV applications deployment.* In 2010 7th IEEE Consumer Communications and Networking Conference, pages 1–2. IEEE, 2010. 32

[Dubois 95] Didier Dubois, Hélène Fargier & Henri Prade. *Fuzzy constraints in job-shop scheduling.* Journal of intelligent Manufacturing, vol. 6, no. 4, pages 215–234, 1995. 53

[Ducatel 01] K. Ducatel, M. Bogdanowicz, F. Scapolo, J. Leijten & J.C. Burgelman. *Scenarios for ambient intelligence in 2010*, 2001. 11

[Edwards 04] George Edwards, Gan Deng, Douglas C. Schmidt, Aniruddha Gokhale & Bala Natarajan. *Model-driven configuration and deployment of component middleware publish/subscribe services.* In of Lecture, 2004. 32

[Eisenhauer 10] Markus Eisenhauer, Peter Rosengren & Pablo Antolin. *Hydra : A development platform for integrating wireless devices and sensors into ambient intelligence systems.* In The Internet of Things, pages 367–373. Springer, 2010. 26

[El Fallah-Seghrouchni 10] Amal El Fallah-Seghrouchni, Andrei Olaru, Nga Thi Thuy Nguyen & Diego Salomone. *Ao Dai : Agent Oriented Design for Ambient Intelligence.* In Nirmit Desai, Alan Liu & Michael Winikoff, editeurs, PRIMA, volume 7057 of *Lecture Notes in Computer Science*, pages 259–269. Springer, 2010. 11, 25

[Emeakaroha 10] Vincent C Emeakaroha, Ivona Brandic, Michael Maurer & Schahram Dustdar. *Low level metrics to high level SLAs-LoM2HiS framework : Bridging the gap between monitored metrics and SLA parameters in cloud environments.* In High Performance Computing and Simulation (HPCS), 2010 International Conference on, pages 48–54. IEEE, 2010. 41

[Emeakaroha 11] Vincent C Emeakaroha, Ivona Brandic, Michael Maurer & Ivan Breskovic. *SLA-Aware application deployment and resource allocation in clouds.* In Computer Software and Applications Conference Workshops (COMPSACW), 2011 IEEE 35th Annual, pages 298–303. IEEE, 2011. 41, 42

[Emeakaroha 13] Vincent C Emeakaroha, Ivona Brandic, Michael Maurer & Schahram Dustdar. *Cloud resource provisioning and SLA enforcement via LoM2HiS framework.* Concurrency and Computation : Practice and Experience, vol. 25, no. 10, pages 1462–1481, 2013. 41

[Evans 11] Dave Evans. *The Internet of Things : How the Next Evolution of the Internet Is Changing Everything*, 2011. 1, 13

[Fan 12a] Pei Fan, Zhenbang Chen, Ji Wang, Zibin Zheng & Michael R Lyu. *Topology-aware deployment of scientific applications in cloud computing.* In Cloud Computing (CLOUD), 2012 IEEE 5th international conference on, pages 319–326. IEEE, 2012. 42

[Fan 12b] Pei Fan, Ji Wang, Zhenbang Chen, Zibin Zheng & Michael R Lyu. *A spectral clustering-based optimal deployment method for scientific application in cloud computing.* International Journal of Web and Grid Services, vol. 8, no. 1, pages 31–55, 2012. 42

[Fard 14] Arash Fard, M Usman Nisar, John A Miller & Lakshmish Ramaswamy. *Distributed and scalable graph pattern matching : Models and algorithms.* International Journal of Big Data (IJBD), vol. 1, no. 1, 2014. 51

[Fasli 07] Maria Fasli. *On agent technology for e-commerce : trust, security and legal issues.* The Knowledge Engineering Review, vol. 22, no. 01, pages 3–35, 2007. 27

[Ferry 13] Nicolas Ferry, Alessandro Rossini, Franck Chauvel, Brice Morin & Arnor Solberg. *Towards model-driven provisioning, deployment, monitoring, and adaptation of multi-cloud systems.* In 2013 IEEE Sixth International Conference on Cloud Computing, pages 887–894. IEEE, 2013. 41, 43

[Flissi 06] Areski Flissi & Philippe Merle. *A generic deployment framework for grid computing and distributed applications.* In OTM Confederated International Conferences" On the Move to Meaningful Internet Systems", pages 1402–1411. Springer, 2006. 38

[Flissi 08] A. Flissi, J. Dubus, N. Dolet & P. Merle. *Deploying on the Grid with DeployWare.* In Cluster Computing and the Grid, 2008. CCGRID '08. 8th IEEE International Symposium on, pages 177–184, May 2008. 38, 115

[Fortino 14] Giancarlo Fortino, Antonio Guerrieri, Wilma Russo & Claudio Savaglio. *Middlewares for smart objects and smart environments : Overview and comparison.* In Internet of Things Based on Smart Objects, pages 1–27. Springer, 2014. 22

[Foster 03] Ian Foster & Carl Kesselman. The grid 2 : Blueprint for a new computing infrastructure. Elsevier, 2003. 36

[Foster 08] Ian Foster, Yong Zhao, Ioan Raicu & Shiyong Lu. *Cloud computing and grid computing 360-degree compared.* In 2008 Grid Computing Environments Workshop, pages 1–10. Ieee, 2008. 39, 40

[Fox 81] Mark S. Fox. *An Organizational View of Distributed Systems*. IEEE Transactions on Systems, Man, and Cybernetics, vol. 11, no. 1, pages 70 – 80, 1981. 131

[Freuder 82] Eugene C Freuder. *A sufficient condition for backtrack-free search*. Journal of the ACM (JACM), vol. 29, no. 1, pages 24–32, 1982. 53

[Gannon 02] Dennis Gannon, Randall Bramley, Geoffrey Fox, Shava Smallen, Al Rossi, Rachana Ananthakrishnan, Felipe Bertrand, Ken Chiu, Matt Farrellee, Madhu Govindaraju*et al.* *Programming the grid : Distributed software components, p2p and grid web services for scientific applications*. Cluster Computing, vol. 5, no. 3, pages 325–336, 2002. 36, 37

[Garey 76] Michael R Garey, David S. Johnson & Larry Stockmeyer. *Some simplified NP-complete graph problems*. Theoretical computer science, vol. 1, no. 3, pages 237–267, 1976. 53

[Garg 10] Saurabh Kumar Garg, Rajkumar Buyya & Howard Jay Siegel. *Time and cost trade-off management for scheduling parallel applications on utility grids*. Future Generation Computer Systems, vol. 26, no. 8, pages 1344–1355, 2010. 41

[Gaschnig 74] John Gaschnig. *A constraint satisfaction method for inference making*. In Proceedings of the Twelfth Annual Allerton Conference on Circuit Systems Theory, pages 866–874, 1974. 53

[Georgantas 10] Nikolaos Georgantas, Valerie Issarny, Sonia Ben Mokhtar, Yerom-David Bromberg, Sebastien Bianco, Graham Thomson, Pierre-Guillaume Raverdy, Aitor Urbieta & Roberto Speicys Cardoso. *Middleware Architecture for Ambient Intelligence in the Networked Home*. In Springer, editeur, Handbook of Ambient Intelligence and Smart Environments, pages 1139–1169. 2010. 11, 26

[Gómez-Goiri 14] Aitor Gómez-Goiri, Pablo Orduña, Javier Diego & Diego López-de Ipiña. *Otsopack : Lightweight semantic framework for interoperable ambient intelligence applications*. Computers in Human Behavior, vol. 30, pages 460–467, 2014. 26

[Hagras 04] Hani Hagras, Victor Callaghan, Martin Colley, Graham Clarke, Anthony Pounds-Cornish & Hakan Duman. *Creating an ambient-intelligence environment using embedded agents*. IEEE Intelligent Systems, vol. 19, no. 6, pages 12–20, 2004. 21

[Haralick 80] Robert M Haralick & Gordon L Elliott. *Increasing tree search efficiency for constraint satisfaction problems*. Artificial intelligence, vol. 14, no. 3, pages 263–313, 1980. 53

[Hays 64] David G Hays. *Dependency theory : A formalism and some observations*. Language, vol. 40, no. 4, pages 511–525, 1964. 49

[Hellenschmidt 04] Michael Hellenschmidt & Thomas Kirste. *A Generic Topology for Ambient Intelligence.* In Panos Markopoulos, Berry Eggen, Emile H. L. Aarts & James L. Crowley, editeurs, EUSAI, volume 3295 of *Lecture Notes in Computer Science*, pages 112–123. Springer, 2004. 11

[Hendler 08] Jim Hendler & Frank van Harmelen. *The Semantic Web : webizing knowledge representation.* Foundations of Artificial Intelligence, vol. 3, pages 821–839, 2008. 49

[Henzinger 95] Monika Rauch Henzinger, Thomas A Henzinger & Peter W Kopke. *Computing simulations on finite and infinite graphs.* In Foundations of Computer Science, 1995. Proceedings., 36th Annual Symposium on, pages 453–462. IEEE, 1995. 51

[Hoffmann 82] Christoph M Hoffmann. *Group-theoretic algorithms and graph isomorphism*, 1982. 51

[Horling 04] Bryan Horling & Victor Lesser. *A Survey of Multi-agent Organizational Paradigms.* Knowl. Eng. Rev., vol. 19, no. 4, pages 281–316, December 2004. 131

[HP 14] HP. *HP Study Reveals 70 Percent of Internet of Things Devices Vulnerable to Attack*, 2014. 143

[Inc 08] SYS-CON Media Inc. *Twenty-One Experts Define Cloud Computing*, 2008. 39

[Islam 12] Sadeka Islam, Kevin Lee, Alan Fekete & Anna Liu. *How a consumer can measure elasticity for cloud platforms.* In Proceedings of the 3rd ACM/SPEC International Conference on Performance Engineering, pages 85–96. ACM, 2012. 41

[ITU-T 12] ITU-T. *Recommendations : Overview of the Internet of things*, 2012. 11

[Jaiswal 05] Ashutosh Jaiswal, Yongdae Kim & Maria Gini. *Design and implementation of a secure multi-agent marketplace.* Electronic Commerce Research and Applications, vol. 3, no. 4, pages 355–368, 2005. 29

[Jennings 15] Brendan Jennings & Rolf Stadler. *Resource management in clouds : Survey and research challenges.* Journal of Network and Systems Management, vol. 23, no. 3, pages 567–619, 2015. 40

[Johanson 02] B. Johanson, A. Fox & Terry Winograd. *The Interactive Workspaces Project : Experiences with Ubiquitous Computing Rooms.* vol. 1, no. 2, 2002. 11

[Juve 10] Gideon Juve, Ewa Deelman, Karan Vahi, Gaurang Mehta, Bruce Berriman, Benjamin P Berman & Phil Maechling. *Data sharing options for scientific workflows on amazon ec2.* In Proceedings of the 2010 ACM/IEEE International Conference for High Performance Computing, Networking,

Storage and Analysis, pages 1–9. IEEE Computer Society, 2010. 40

[Juve 11a] Gideon Juve & Ewa Deelman. *Automating application deployment in infrastructure clouds.* In Cloud Computing Technology and Science (CloudCom), 2011 IEEE Third International Conference on, pages 658–665. IEEE, 2011. 40, 42, 43

[Juve 11b] Gideon Juve & Ewa Deelman. *Wrangler : virtual cluster provisioning for the cloud.* In Proceedings of the 20th international symposium on High performance distributed computing, pages 277–278. ACM, 2011. 42

[Kameas 10] Achilles Kameas & Ioannis Calemis. *Pervasive systems in health care.* In Handbook of ambient intelligence and smart environments, pages 315–346. Springer, 2010. 20

[Kanies 06] Luke Kanies. *Puppet : Next-generation configuration management.* ; login : : the magazine of USENIX & SAGE, vol. 31, no. 1, pages 19–25, 2006. 31, 41

[Kannan 09] Kalapriya Kannan, Nanjangud C Narendra & Lakshmish Ramaswamy. *Managing configuration complexity during deployment and maintenance of SOA solutions.* In Services Computing, 2009. SCC'09. IEEE International Conference on, pages 152–159. IEEE, 2009. 35

[Keahey 09] Katarzyna Keahey, Mauricio Tsugawa, Andrea Matsunaga & Jose Fortes. *Sky computing.* IEEE Internet Computing, vol. 13, no. 5, pages 43–51, 2009. 41

[Kichkaylo 04] Tatiana Kichkaylo & Vijay Karamcheti. *Optimal resource-aware deployment planning for component-based distributed applications.* In High performance Distributed Computing, 2004. Proceedings. 13th IEEE International Symposium on, pages 150–159. IEEE, 2004. 33

[Kim 91] Whoi-Yul Kim & Avinash C Kak. *3-D object recognition using bipartite matching embedded in discrete relaxation.* IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 13, no. 3, pages 224–251, 1991. 50

[Kirchbuchner 15] Florian Kirchbuchner, Tobias Grosse-Puppendahl, Matthias R. Hastall, Martin Distler & Arjan Kuijper. Ambient intelligence : 12th european conference, ami 2015, athens, greece, november 11-13, 2015, proceedings, chapitre Ambient Intelligence from Senior Citizens' Perspectives : Understanding Privacy Concerns, Technology Acceptance, and Expectations, pages 48–59. Springer International Publishing, 2015. 16

[Koestler 67] Arthur Koestler. The ghost in the machine. Hutchinson, 1967. 131

[Kravari 15] Kalliopi Kravari & Nick Bassiliades. *A Survey of Agent Platforms*. Journal of Artificial Societies and Social Simulation, vol. 18, no. 1, page 11, 2015. 146

[Krupa 10] Yann Krupa & Laurent Vercouter. *Contextual Integrity and Privacy Enforcing Norms for Virtual Communities.* In Olivier Boissier, Amal El Fallah Seghrouchni, Salima Hassas & Nicolas Maudet, editeurs, MALLOW, volume 627 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2010. 28

[Kumar 92] Vipin Kumar. *Algorithms for constraint-satisfaction problems : A survey.* AI magazine, vol. 13, no. 1, page 32, 1992. 53

[Lacour 04a] Sébastien Lacour, Christian Pérez & Thierry Priol. *A network topology description model for grid application deployment.* In Proceedings of the 5th IEEE/ACM International Workshop on Grid Computing, pages 61–68. IEEE Computer Society, 2004. 38

[Lacour 04b] Sébastien Lacour, Christian Pérez & Thierry Priol. *A software architecture for automatic deployment of CORBA components using grid technologies.* arXiv preprint cs/0411086, 2004. 38

[Lacour 05] Sébastien Lacour, Christian Pérez & Thierry Priol. *Generic application description model : Toward automatic deployment of applications on computational grids.* In Proceedings of the 6th IEEE/ACM International Workshop on Grid Computing, pages 284–287. IEEE Computer Society, 2005. 38

[Larrosa 04] Javier Larrosa & Gabriel Valiente. *Constraint satisfaction algorithms for graph pattern matching.* Mathematical Structures in Computer Science, vol. 12, pages 403–422, 8 2004. 50, 58

[Lech 05] Till C Lech & Leendert WM Wienhofen. *AmbieAgents : a scalable infrastructure for mobile and context-aware information services.* In Proceedings of the fourth international joint conference on Autonomous agents and multiagent systems, pages 625–631. ACM, 2005. 23

[Lee 10] Young Choon Lee, Chen Wang, Albert Y Zomaya & Bing Bing Zhou. *Profit-driven service request scheduling in clouds.* In Proceedings of the 2010 10th IEEE/ACM international conference on cluster, cloud and grid computing, pages 15–24. IEEE Computer Society, 2010. 41

[Li 15] Shancang Li, Li Da Xu & Shanshan Zhao. *The internet of things : a survey.* Information Systems Frontiers, vol. 17, no. 2, pages 243–259, 2015. 12

[Ma 11] Shuai Ma, Yang Cao, Wenfei Fan, Jinpeng Huai & Tianyu Wo. *Capturing topology in graph pattern matching.*

Proceedings of the VLDB Endowment, vol. 5, no. 4, pages 310–321, 2011. 51

[Ma 12] Shuai Ma, Yang Cao, Jinpeng Huai & Tianyu Wo. *Distributed graph pattern matching.* In Proceedings of the 21st international conference on World Wide Web, pages 949–958. ACM, 2012. 51

[Mabrouki 14] Olfa Mabrouki. *Semantic Framework for Managing Privacy Policies in Ambient Intelligence.* PhD thesis, Université Paris Sud-Paris XI, 2014. 27

[Mackworth 77] Alan K Mackworth. *Consistency in networks of relations.* Artificial intelligence, vol. 8, no. 1, pages 99–118, 1977. 53

[Marples 01] Dave Marples & Peter Kriens. *The open services gateway initiative : An introductory overview.* IEEE Communications Magazine, vol. 39, no. 12, pages 110–114, 2001. 32

[Marshall 10] Paul Marshall, Kate Keahey & Tim Freeman. *Elastic site : Using clouds to elastically extend site resources.* In Proceedings of the 2010 10th IEEE/ACM International Conference on Cluster, Cloud and Grid Computing, pages 43–52. IEEE Computer Society, 2010. 41

[Masterman 61] Margaret Masterman. *Semantic message detection for machine translation, using an interlingua.* In Proc. 1961 International Conf. on Machine Translation, pages 438–475, 1961. 49

[McBride 04] Brian McBride. *The resource description framework (RDF) and its vocabulary description language RDFS.* In Handbook on ontologies, pages 51–65. Springer, 2004. 54

[McGuinness 04] Deborah L McGuinness, Frank Van Harmelen*et al.* *OWL web ontology language overview.* W3C recommendation, vol. 10, no. 10, page 2004, 2004. 57

[McKay 14] Brendan D McKay & Adolfo Piperno. *Practical graph isomorphism, II.* Journal of Symbolic Computation, vol. 60, pages 94–112, 2014. 50

[Menczer 02] F Menczer, WN Street, N Vishwakarma, A Monge & M Jakobsson. *IntelliShopper : A Proactive, Personal, Private Shopping Assistant.* In Proc. 1st ACM Int. Joint Conf. on Autonomous Agents and MultiAgent Systems (AAMAS), 2002. 29

[Messmer 00] Bruno T. Messmer & Horst Bunke. *Efficient Subgraph Isomorphism Detection : A Decomposition Approach.* IEEE Trans. on Knowl. and Data Eng., vol. 12, no. 2, March 2000. 50

[Minton 92] Steven Minton, Mark D Johnston, Andrew B Philips & Philip Laird. *Minimizing conflicts : a heuristic repair method for constraint satisfaction and scheduling problems.* Artificial Intelligence, vol. 58, no. 1-3, pages 161–205, 1992. 53

[Montanari 74] Ugo Montanari. *Networks of constraints : Fundamental properties and applications to picture processing.* Information sciences, vol. 7, pages 95–132, 1974. 53

[Mozer 98] Michael C Mozer. *The neural network house : An environment hat adapts to its inhabitants.* In Proc. AAAI Spring Symp. Intelligent Environments, volume 58, 1998. 20

[Mozer 04] Michael Mozer. *Lessons from an adaptive house.* PhD thesis, Architectural Engineering, 2004. 20

[Myrhaug 01] Hans Inge Myrhaug. *Towards life-long and personal context spaces.* In Workshop on User Modelling for Context-Aware Applications, 2001. 23

[Nathan 11] Marz Nathan. *Storm : Distributed and fault-tolerant realtime computation*, 2011. 15

[Noack 09] Andreas Noack & Randolf Rotta. *Multi-level algorithms for modularity clustering.* In International Symposium on Experimental Algorithms, pages 257–268. Springer, 2009. 42

[O'Hare 12] Gregory M. P. O'Hare, Rem Collier, Mauro Dragone, Michael J. O'Grady, Conor Muldoon & Alcides de J. Montoya. *Embedding Agents within Ambient Intelligent Applications.* In Tibor Bosse, editeur, Agents and Ambient Intelligence, volume 12 of *Ambient Intelligence and Smart Environments*, pages 119–133. IOS Press, 2012. 11

[Olaru 11] Andrei Olaru. *A context-aware multi-agent system for AmI environments.* Teza de doctorat, 2011. 161

[Olaru 13] Andrei Olaru, Adina Magda Florea & Amal El Fallah Seghrouchni. *A context-aware multi-agent system as a middleware for ambient intelligence.* Mobile Networks and Applications, vol. 18, no. 3, pages 429–443, 2013. 25

[Olaru 15] Andrei Olaru, Marius-Tudor Benea, Amal El Fallah Seghrouchni & Adina Magda Florea. *tATAmI : A platform for the development and deployment of agent-based ami applications.* Procedia Computer Science, vol. 52, pages 476–483, 2015. 25

[Pandey 10] Suraj Pandey, Linlin Wu, Siddeswara Mayura Guru & Rajkumar Buyya. *A particle swarm optimization-based heuristic for scheduling workflow applications in cloud computing environments.* In 2010 24th IEEE international conference on advanced information networking and applications, pages 400–407. IEEE, 2010. 41

[Papadopoulos 03] Philip M Papadopoulos, Mason J Katz & Greg Bruno. *NPACI Rocks : Tools and techniques for easily deploying manageable linux clusters.* Concurrency and Computation : Practice and Experience, vol. 15, no. 7-8, pages 707–725, 2003. 31, 41

[Peirce 80] Charles Sanders Peirce. *On the algebra of logic.* American Journal of Mathematics, vol. 3, no. 1, pages 15–57, 1880. 52

[Peirce 85] Charles Sanders Peirce. *On the algebra of logic : A contribution to the philosophy of notation.* American journal of mathematics, vol. 7, no. 2, pages 180–196, 1885. 52

[Pérez 03] Christian Pérez, Thierry Priol & André Ribes. *A parallel CORBA component model for numerical code coupling.* International Journal of High Performance Computing Applications, vol. 17, no. 4, pages 417–429, 2003. 37

[Petcu 05] Adrian Petcu & Boi Faltings. *A scalable method for multiagent constraint optimization.* Rapport technique, 2005. 53

[Petcu 06] Adrian Petcu & Boi Faltings. *ODPOP : An algorithm for open/distributed constraint optimization.* In Proceedings of the National Conference on Artificial Intelligence, volume 21, page 703. Menlo Park, CA ; Cambridge, MA ; London ; AAAI Press ; MIT Press ; 1999, 2006. 53

[Petcu 10] Dana Petcu, Ciprian Crăciun, Marian Neagul, Silviu Panica, Beniamino Di Martino, Salvatore Venticinque, Massimiliano Rak & Rocco Aversa. *Architecturing a sky computing platform.* In European Conference on a Service-Based Internet, pages 1–13. Springer, 2010. 41

[Pretz 13] Kathy Pretz. *The next evolution of the internet.* IEEE Magazine The institute, vol. 50, no. 5, 2013. 12

[Ramchurn 04] Sarvapali D. Ramchurn, Dong Huynh & Nicholas R. Jennings. *Trust in Multi-agent Systems.* Knowl. Eng. Rev., vol. 19, no. 1, pages 1–25, March 2004. 29

[Ramos 10] Carlos Ramos, Goreti Marreiros, Ricardo Santos & Carlos Filipe Freitas. *Smart offices and intelligent decision rooms.* In Handbook of Ambient Intelligence and Smart Environments, pages 851–880. Springer, 2010. 20

[Rao 04] Jinghai Rao & Xiaomeng Su. *A survey of automated web service composition methods.* In International Workshop on Semantic Web Services and Web Process Composition, pages 43–54. Springer, 2004. 33

[Redlin 06] C Redlin & K Carlson-Neumann. *Websphere process server and websphere enterprise service bus deployment patterns.* Rapport technique, Technical report, IBM (November 2006), 2006. 34

[Ricci 03] Alessandro Ricci. *Agents and Coordination Artifacts for Feature Engineering.* In Mark Dermot Ryan, John-Jules Ch. Meyer & Hans-Dieter Ehrich, editeurs, Objects, Agents, and Features, volume 2975 of *Lecture Notes in Computer Science*, pages 209–226. Springer, 2003. 115

[Ricci 11] Alessandro Ricci, Michele Piunti & Mirko Viroli. *Environment programming in multi-agent systems : an artifact-based perspective.* vol. 23, no. 2, pages 158–192, 2011. 115

[Rich 07] CR Rich. *TADDM's flexible approach to discovery.* The Tivoli Advisor, Technical Report, 2007. 35

[Richens 56] Richard H Richens. *Preprogramming for mechanical translation.* Mechanical Translation, vol. 3, no. 1, pages 20–25, 1956. 49

[Rivera 11] Hanessa Rivera & Rob van der Meulen. *Gartner Says 4.9 Billion Connected "Things" Will Be in Use in 2015*, 2011. 13

[Rodríguez 08] Marcela D Rodríguez & Jesús Favela. *An agent middleware for ubiquitous computing in healthcare.* In Advanced Computational Intelligence Paradigms in Healthcare-3, pages 117–149. Springer, 2008. 27

[Sabater 05] Jordi Sabater & Carles Sierra. *Review on computational trust and reputation models.* Artificial intelligence review, vol. 24, no. 1, pages 33–60, 2005. 29

[Sacramento 04] Vagner Sacramento, Markus Endler, Hana K Rubinsztejn, Luciana S Lima, Kleder Goncalves, Fernando N Nascimento & Giulliano Almeida Bueno. *MoCA : A middleware for developing collaborative applications for mobile users.* IEEE Distributed systems online, vol. 5, no. 10, pages 2–2, 2004. 25

[Sadeh 05] Norman M Sadeh, Fabien L Gandon & Oh B Kwon. *Ambient intelligence : The mycampus experience.* Rapport technique, DTIC Document, 2005. 21, 27

[Sapuntzakis 03] Constantine P Sapuntzakis, David Brumley, Ramesh Chandra, Nickolai Zeldovich, Jim Chow, Monica S Lam & Mendel Rosenblum. *Virtual Appliances for Deploying and Maintaining Software.* In LISA, volume 3, pages 181–194, 2003. 30

[Satoh 00] Ichiro Satoh. *MobileSpaces : A framework for building adaptive distributed applications using a hierarchical mobile agent system.* In Distributed Computing Systems, 2000. Proceedings. 20th International Conference on, pages 161–168. IEEE, 2000. 23

[Satoh 04] Ichiro Satoh. *Mobile agents for ambient intelligence.* In International Workshop on Massively Multiagent Systems, pages 187–201. Springer, 2004. 22

[Schantz 02] Richard E Schantz & Douglas C Schmidt. *Middleware for distributed systems : Evolving the common structure for network-centric applications.* Encyclopedia of Software Engineering, vol. 1, 2002. 31

[Schiele 10] Gregor Schiele, Marcus Handte & Christian Becker. *Pervasive computing middleware.* In Handbook of Ambient Intelligence and Smart Environments, pages 201–227. Springer, 2010. 22

[Schiex 92] Thomas Schiex. *Possibilistic constraint satisfaction problems or How to handle soft constraints ?* In Proceedings of the Eighth international conference on Uncertainty in artificial intelligence, pages 268–275. Morgan Kaufmann Publishers Inc., 1992. 53

[Seghrouchni 08] Amal El Fallah Seghrouchni, Karin Breitman, Nicolas Sabouret, Markus Endler, Yasmine Charif & Jean-Pierre Briot. *Ambient intelligence applications : Introducing the campus framework.* In Engineering of Complex Computer Systems, 2008. ICECCS 2008. 13th IEEE International Conference on, pages 165–174. IEEE, 2008. 24

[Selman 92] Bart Selman, Hector J Levesque, David G Mitchell*et al. A New Method for Solving Hard Satisfiability Problems.* In AAAI, volume 92, pages 440–446, 1992. 53

[Sharma 11] Upendra Sharma, Prashant Shenoy, Sambit Sahu & Anees Shaikh. *A cost-aware elasticity provisioning system for the cloud.* In Distributed Computing Systems (ICDCS), 2011 31st International Conference on, pages 559–570. IEEE, 2011. 41

[Shen 11] Zhiming Shen, Sethuraman Subbiah, Xiaohui Gu & John Wilkes. *Cloudscale : elastic resource scaling for multi-tenant cloud systems.* In Proceedings of the 2nd ACM Symposium on Cloud Computing, page 5. ACM, 2011. 41

[Solove 06] Daniel J Solove. *A taxonomy of privacy.* University of Pennsylvania law review, pages 477–564, 2006. 27

[Sowa 76] John F Sowa. *Conceptual graphs for a data base interface.* IBM Journal of Research and Development, vol. 20, no. 4, pages 336–357, 1976. 52

[Sowa 83] John F Sowa. *Conceptual structures : information processing in mind and machine.* 1983. 52

[Stajano 10] Frank Stajano. *Security issues in ubiquitous computing.* In Handbook of Ambient Intelligence and Smart Environments, pages 281–314. Springer, 2010. 26

[Stavropoulos 12] Thanos G Stavropoulos, Dimitris Vrakas, Danai Vlachava & Nick Bassiliades. *Bonsai : a smart building ontology for ambient intelligence.* In Proceedings of the 2nd international conference on web intelligence, mining and semantics, page 30. ACM, 2012. 26

[Stavropoulos 13] Thanos G Stavropoulos, Konstantinos Gottis, Dimitris Vrakas & Ioannis Vlahavas. *aWESoME : A web service middleware for ambient intelligence.* Expert Systems with Applications, vol. 40, no. 11, pages 4380–4392, 2013. 11, 26

[Stutzle 05] T Stutzle & H Hoos. *Stochastic Local Search : Foundations and Applications*, 2005. 53

[Such 12] Jose M. Such, Agustín Espinosa, Ana GarcíA-Fornes & Carles Sierra. *Self-disclosure Decision Making Based on Intimacy and Privacy.* Inf. Sci., vol. 211, pages 93–111, November 2012. 29

[Such 14] Jose M. Such, Agustín Espinosa & Ana García-Fornes. *A survey of privacy in multi-agent systems.* The Knowledge Engineering Review, vol. 29, pages 314–344, 6 2014. 27

[Swan 13] Melanie Swan. *The quantified self : Fundamental disruption in big data science and biological discovery.* Big Data, vol. 1, no. 2, pages 85–99, 2013. 14

[Tentori 06] Monica Tentori, Jesus Favela & Marcela D. Rodriguez. *Privacy-Aware Autonomous Agents for Pervasive Healthcare.* IEEE Intelligent Systems, vol. 21, no. 6, pages 55–62, November 2006. 27

[Tesnière 59] Lucien Tesnière. *Eléments de Syntaxe Structurale*, 1959. 49

[Tsai 79] Wen-Hsiang Tsai & King-Sun Fu. *Error-correcting isomorphisms of attributed relational graphs for pattern analysis.* IEEE Transactions on systems, man, and cybernetics, vol. 9, no. 12, pages 757–768, 1979. 50

[Udupi 10] Yathiraj B. Udupi & Munindar P. Singh. *Agents and Peer-to-Peer Computing.* chapitre Information Sharing Among Autonomous Agents in Referral Networks, pages 13–26. Springer-Verlag, 2010. 28

[Ullmann 76] Julian R. Ullmann. *An Algorithm for Subgraph Isomorphism.* J. ACM, vol. 23, no. 1, pages 31–42, 1976. 50

[Van Harmelen 08] Frank Van Harmelen, Vladimir Lifschitz & Bruce Porter. Handbook of knowledge representation, volume 1. Elsevier, 2008. 45

[Vanneschi 02] Marco Vanneschi. *The programming model of ASSIST, an environment for parallel and distributed portable applications.* Parallel computing, vol. 28, no. 12, pages 1709–1732, 2002. 37

[Vinoski 97] Steve Vinoski. *CORBA : integrating diverse applications within distributed heterogeneous environments.* IEEE Communications magazine, vol. 35, no. 2, pages 46–55, 1997. 32, 37

[Wei 10] Yi Wei & M Brian Blake. *Service-oriented computing and cloud computing : challenges and opportunities.* IEEE Internet Computing, vol. 14, no. 6, page 72, 2010. 44

[Weiser 93] Mark Weiser. *Some Computer Science Problems in Ubiquitous Computing.* vol. 36, no. 7, pages 74–84, 1993. 10

[Westin 67] Alan F. Westin. Privacy and Freedom. Atheneum, New York, 1967. 27

[White 12] Tom White. Hadoop, the definitive guide. O'Reilly, 2012. 15

[Wolf 10] Gary Wolf, A Carmichael & K Kelly. *The quantified self.* http ://www.ted.com/talks/gary_wolf_the_quantified_self.html, 2010. 14

[Woods 75] WA Woods. *Whats in a link : Foundations for semantic networks.* 1975. 51

[Yokoo 92] Makoto Yokoo, Toru Ishida, Edmund H Durfee & Kazuhiro Kuwabara. *Distributed constraint satisfaction for formalizing distributed problem solving.* In Distributed Computing Systems, 1992., Proceedings of the 12th International Conference on, pages 614–621. IEEE, 1992. 53

[Yokoo 94] Makoto Yokoo. *Weak-commitment search for solving constraint satisfaction problems.* In AAAI, volume 94, pages 313–318, 1994. 53

[Yokoo 12] Makoto Yokoo. Distributed constraint satisfaction : foundations of cooperation in multi-agent systems. Springer Science & Business Media, 2012. 53

[Zhou 10] Minqi Zhou, Rong Zhang, Wei Xie, Weining Qian & Aoying Zhou. *Security and privacy in cloud computing : A survey.* In Semantics Knowledge and Grid (SKG), 2010 Sixth International Conference on, pages 105–112. IEEE, 2010. 16