Réduire la complexité : une problématique centrale de la construction des systèmes répartis

Sébastien Leriche

Mémoire d'Habilitation à Diriger des Recherches

Délivré par : *l'Institut National Polytechnique de Toulouse (INP Toulouse)*

Présentée et soutenue le *25/11/2016* **par :**

Sébastien LERICHE

Réduire la complexité : une problématique centrale de la construction des systèmes répartis

JURY

Daniel HAGIMONT (Prof. HDR)	INP/IRIT/ENSEEIHT, Toulouse	Corr. recherche
Olivier BARAIS (Prof. HDR)	Université de Rennes 1/IRISA/INRIA/B-COM, Rennes	Rapporteur
Noël DE PALMA (Prof. HDR)	IMAG/LIG, Grenoble	Rapporteur
Michel RIVEILL (Prof. HDR)	École Polytechnique de l'Université Nice Sophia Antipolis, Nice	Rapporteur
Philippe ROOSE (MdC HDR)	LIUPPA Lab/IUT Bayonne/Université de Pau, Anglet	Examinateur

École doctorale et spécialité :
EDSYS : Informatique 4200018
Unité de Recherche :
ENAC - Laboratoire d'Informatique Interactive
Correspondant recherche INP :
Daniel HAGIMONT
Rapporteurs :
Olivier BARAIS, *Noël DE PALMA* et *Michel RIVEILL*

Remerciements

Je tiens à remercier chaleureusement les membres du jury et tout d'abord mes trois rapporteurs : Olivier BARAIS, Noël DE PALMA et Michel RIVEILL, pour leur attention passée à la lecture de ce document et leurs remarques pertinentes et encourageantes. La dernière version de ce rapport a été positivement enrichie grâce aux commentaires de Daniel HAGIMONT, mon correspondant recherche à l'INP, qui m'a été aussi d'une grande aide dans les méandres administratifs et grâce à qui j'ai pu atteindre mon objectif dans les temps. Enfin, Philippe ROOSE m'a fait le plaisir d'accepter de compléter le jury en tant qu'examinateur, ses encouragements lorsque nous nous sommes croisés au détours de conférences ont portés leurs fruits.

Le travail exposé dans ce mémoire a été rendu possible grâce aux activités de recherche auxquelles ont également participé un grand nombre de personnes que je souhaite remercier, en particulier à l'Université Paul Sabatier (Jean-Paul, Christine, Frédéric) à Télécom SudParis (Chantal, Claire, Denis, Michel, Sophie) à l'ENAC (les Stéphane, les Mathieu, Daniel, Hélène, Jean-Luc, Raïlane, Stéphanie et Catherine). Et surtout merci à mes trois doctorants que j'ai eu le plaisir d'encadrer entre Evry et Toulouse : Amine, Raja et Sam. Parce qu'après tout, une partie de ce travail est fondamentalement une partie de leur travail.

Table des matières

1 Introduction

Ce manuscrit présente une synthèse de mes contributions dans le domaine de la recherche depuis 2006, durant mon activité d'enseignant-chercheur qui s'est déroulée successivement à Toulouse (IRIT/UPS) puis à Évry (TélécomSudParis/UMR SAMOVAR) pendant presque sept ans pour revenir à Toulouse (ENAC) où je suis actuellement employé depuis 2013. Conformément aux exigences de l'exercice, il contient également dans la dernière partie un projet de recherche original, dans la continuité de mes travaux et adapté au contexte applicatif de mon équipe de recherche.

Le domaine des systèmes répartis m'a fasciné continuellement depuis que je l'ai découvert. Les possibilités offertes par l'interconnexion des appareils, la combinatoire des ressources et des services, laissaient imaginer des évolutions majeures dans les usages de l'informatique, au quotidien des utilisateurs, mais aussi au service des entreprises innovantes. C'était il y a vingt ans environ, au moment de cette période euphorique de croissance de la bulle Internet[1]. C'est aussi la période où Weiser prophétise un changement majeur dans les équipements connectés et invente le concept d'informatique ubiquitaire. « *The most profound technologies are those that disappear. They weave themselves into the fabric of everyday life until they are indistinguishable from it.* » [Weiser, 1991]. Les utilisateurs ne se serviront pas seulement d'un ordinateur, mais les technologies environnantes serviront l'utilisateur. Cette prophétie semble accomplie aujourd'hui, l'informatique ambiante est une réalité, décrite depuis plusieurs années par Mattern [Mattern, 2001] ou Waldner par exemple [Waldner, 2007].

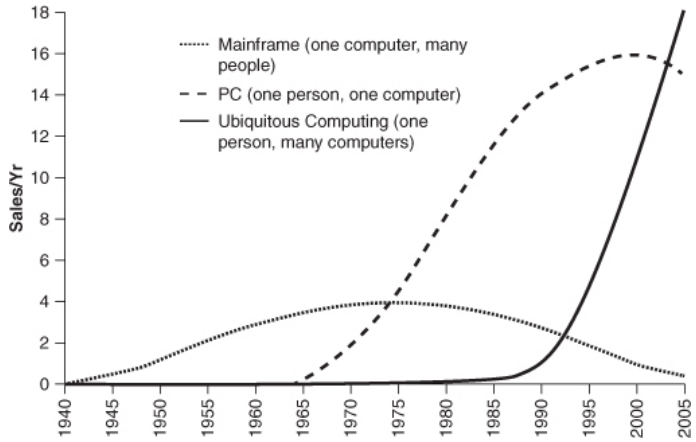

Figure 1.1 – Les tendances majeures de l'Informatique [Weiser, 1996]

1. https://fr.wikipedia.org/wiki/Bulle_Internet

Cette évolution a suscité l'élaboration de nombreux programmes de recherche tout autour de la planète pour mieux comprendre les enjeux, les problématiques et tenter de répondre à certaines questions. Au-delà des usages et des impacts sociétaux, les questions techniques les plus importantes tournent autour de la soutenabilité et la résilience des systèmes ainsi que de la sécurité. Parmi tous ces traits et durant ces dix dernières années, je me suis penché sur une question centrale, celle de la complexité de ces systèmes et avec un objectif derrière tous mes travaux, celui de réduire cette complexité.

1.1 Évolution des thèmes de recherche

Sur le plan de l'évolution des système, la période des années 2000 est celle de la mise en œuvre d'architectures réparties à grande échelle. Le terme de grille de calcul est proposé en 1998 par Foster [Foster and Kesselman, 1998], par analogie au réseau électrique, comme agglomérat de ressources (physiques ou logiques) interconnectées par l'Internet et dont l'hétérogénéité est masquée par une interface d'accès [Foster et al., 2001, Juhász et al., 2004]. Le concept de grille a eu principalement du succès dans les secteurs nécessitant une forte puissance de calcul (scientifique, financier) mais peu d'usage sous cette appellation pour le grand public. Celui-ci est touché par l'arrivée massive du pair-à-pair (P2P) [Milojicic et al., 2002]. Cette architecture permet de s'affranchir d'un serveur centralisé au moyen d'un logiciel spécifiquement conçu ou d'une couche middleware dédiée [Ratnasamy et al., 2001, Stoica et al., 2001].

Par la suite, l'émergence du paradigme de l'Internet des objets ou des choses (*Internet of Things, IoT*) [Chen, 2012, Miorandi et al., 2012, Gubbi et al., 2013] place l'utilisateur au cœur d'un environnement intelligent et connecté à l'Internet. En associant ce paradigme aux grandes capacités de stockage et de calculs offertes par les paradigmes de la grille (*Grid Computing*) [Foster and Kesselman, 1998] ou de l'informatique en nuage (*Cloud Computing*) [Dikaiakos et al., 2009, Mell and Grance, 2011], ainsi qu'à des systèmes intermédiaires, comme les appareils mobiles [Satyanarayanan, 2011] ou les serveurs de proximité [Satyanarayanan et al., 2009], les possibilités de conception de systèmes répartis distribués sont augmentées.

Pour concevoir les systèmes logiciels au dessus de ces infrastructures, le concept de composant logiciel, en tant qu'évolution de celui d'objet, a été mis en avant à partir des années 2000 [Szyperski, 2002, Oussalah, 2005]. Son utilisation concerne plus particulièrement le domaine du génie logiciel : la programmation, la vérification ou les infrastructures logicielles. Des applications réussies ont été conduites dans les domaines du commerce électronique, des systèmes d'exploitation, des télécommunications et des jeux vidéo. L'absence de consensus sur une définition claire du *composant logiciel* a permis un foisonnement de travaux de recherche. Les chercheurs s'accordent sur l'unité de composition d'une application, mais pas systématiquement sur sa granularité ou son unicité en tant que concept structurant.

Le domaine des architectures logicielles propose de décrire la structure d'un logiciel comme un assemblage, ou plutôt une interconnexion, de composants. Dans cette optique, le critère de neutralité associé à la notion de composant est très fort : il n'est fait initialement aucune supposition sur le rôle d'un composant [Garlan and Shaw, 1994].

Le composant est par ailleurs utilisé comme un excellent support d'adaptation [Dowling and Cahill, 2001]. Principalement grâce à la possibilité de le découpler, il peut être alors modifié puis remplacé, de manière statique ou dynamique. La problématique du

déploiement découle directement de l'application directe des capacités d'adaptation statique : de nombreux auteurs cherchent les moyens de structurer des applications puis de les déployer en utilisant des composants [DECOR 04, 2004].

Parallèlement, issus du domaine de l'intelligence artificielle, les systèmes multi-agents proposent des modèles, des technologies et des méthodes répondant à des problèmes scientifiques qui concernent en particulier l'ingénierie des systèmes en contribuant au développement de l'infrastructure de communication et de coopération pour des systèmes complexes, décentralisés et ouverts [Ferber, 1995, Briot, 2004].

Des besoins croisés sont mis en évidence dans différents travaux. Ainsi, par exemple, la notion de composant est utilisée pour aider à la conception, la construction et le déploiement d'agents ou de systèmes multi-agents, modulaires et réutilisables. Inversement d'autres tentatives cherchent à transposer des propriétés des agents vers les composants pour concevoir les applications réparties du futur. Dans ce cas il s'agit de donner plus d'autonomie aux composants de l'application : capacités d'adaptation, de prise de décision, d'auto-assemblage (incluant le match-making), de coordination (avec d'autres composants), et donc de s'inspirer de concepts (autonomie, adaptation, coordination) développés et mis en œuvre au sein des systèmes multi-agents. Ces différents aspects sont analysés en détail dans [Briot, 2014] et une de mes contribution à cette analyse est dans [Arcangeli et al., 2006].

La problématique du déploiement est mise en avant dès les premières réalisations de systèmes répartis complexes [Carzaniga et al., 1998a]. Le déploiement est une opération, ou un ensemble d'opérations, qui suit la réalisation d'un logiciel. Le processus de déploiement a pour objectif de rendre ce logiciel disponible pour l'utilisation, puis de le maintenir dans un état opérationnel. C'est un processus complexe qui comprend un certain nombre d'activités liées [Dearle, 2007a].

Dans le domaine du déploiement, les problématiques principalement visées sont la prise en compte de l'hétérogénéité (Disnix [van der Burg and Dolstra, 2011], DeployWare [Flissi et al., 2008a], TUNe [Broto et al., 2008, Toure et al., 2010], SmartFrog [Sabharwal, 2006, Goldsack et al., 2009]), la limitation des ressources de la machine (Kalimucho [Louberry et al., 2011a], Cloudlet [Satyanarayanan et al., 2009]), et la dynamique du domaine, c'est-à-dire les connexions et les déconnexions (spécialement pour les réseaux spontanés), les pannes de machines et des liens de communication, les variations de qualité de services, etc. À part pour TUNe, aucune de ces propositions ne gère l'hétérogénéité, le passage à l'échelle et la dynamique en même temps.

1.2 Problématique et positionnement scientifique

Un "système complexe" [Chavalarias et al., 2009] peut qualifier n'importe quel système composé d'un grand nombre d'entités hétérogènes, dans lequel des interactions locales entre entités créent différents niveaux de structures et d'organisations collectives.

Les progrès constants dans le domaine des Technologies de l'Information et de la Communication (TIC) permettent aux concepteurs de systèmes répartis d'imaginer des systèmes innovants qui répondent à des besoins de plus en plus variés.

Cependant, la grande diversité des technologies existantes dans de tels systèmes implique une augmentation de la complexité du travail de conception, de développement et de déploiement, en augmentant l'hétérogénéité des infrastructures sur lesquelles ils sont

construits [Blair and Grace, 2012, van Steen et al., 2012]. De plus, ces systèmes peuvent être utilisés par des utilisateurs dispersés géographiquement et dont l'organisation sociale est également complexe.

Les principales caractéristiques des systèmes répartis complexes sont le grand nombre d'entités, la forte hétérogénéité technique, leur architecture dynamique, et les différents niveaux d'interactions entre entités. Nos travaux, de nature *génie logiciel* visent deux grandes phases du cycle de vie du logiciel : la production et le déploiement.

Dans les travaux à la continuité de ma thèse de doctorat, nous avons étudié des techniques et des outils (modèles, architectures, *middlewares*, *frameworks*...) qui contribuent à limiter la complexité induite par la prise en compte des problèmes (communs) posés par la grille et l'informatique ambiante : hétérogénéité, volatilité des ressources, volume des données échangées. En particulier, nous avons proposé un patron de conception à base de composants logiciels et d'agents logiciels pour structurer les applications de type P2P [Androutsellis-Theotokis and Spinellis, 2004] complété par une architecture d'agent flexible permettant la prise en compte de propriétés extra-fonctionnelles spécifiques lors de la conception.

Dans les travaux liés à la thèse de Sam, nous avons distingué le concept de répartition multiéchelle de celui plus traditionnel de la répartition à large (ou grande) échelle. Pour nous, le terme large échelle a un sens quantitatif (grand nombre d'équipements, d'utilisateurs, de données à traiter, etc.), alors que le terme multiéchelle exprime une forte hétérogénéité. L'hétérogénéité du système peut venir de différences de latences ou de protocoles des réseaux impliqués, de différences d'espace de stockage ou de nature des équipements, ou de variations de la dispersion des entités. Nous proposons d'utiliser la notion d'échelle comme cadre d'étude pour obtenir une vision simplifiée d'un système réparti complexe. Nous proposons d'appliquer cette démarche aux systèmes répartis complexes qualifiés de multiéchelles en identifiant différentes échelles pertinentes dans ces systèmes. Nous pensons que, dans le domaine de l'informatique, les échelles présentes dans un système réparti complexe peuvent être vues comme des ensembles cohérents et homogènes qui peuvent être associés à des protocoles de communications, des architectures ou des comportements propres.

Avec Amine puis autour du projet ANR INCOME et de la thèse de Raja, nous avons considéré la phase de déploiement. Du fait de la distribution et de l'ubiquité, du nombre et de l'hétérogénéité, de la mobilité et de l'évolutivité des systèmes logiciels, et plus généralement de la dynamique et de la complexité des structures matérielles et logicielles, le déploiement demande de l'automatisation et de l'autonomie. En particulier, le déploiement des systèmes multi-échelles est trop complexe à réaliser de manière efficace par un opérateur humain. Il est donc nécessaire de le penser d'une manière différente des approches traditionnelles. Il doit répondre à des exigences et à des contraintes émanant de différentes parties prenantes et portant à la fois sur le logiciel à déployer et les machines cibles, en particulier sur leur distribution et leur dynamique. Concernant la mobilité et l'ouverture, le déploiement doit réagir à l'instabilité du réseau de machines *c.-à-d.* aux connexions et déconnexions et aux variations de la disponibilité et de la qualité des ressources. Ainsi, le déploiement multi-échelle doit être un processus continu qui supporte l'exécution et nécessite une certaine adaptabilité.

1.3 Rapport d'activités

La figure 1.2 présente une synthèse chronologique de mes activités.

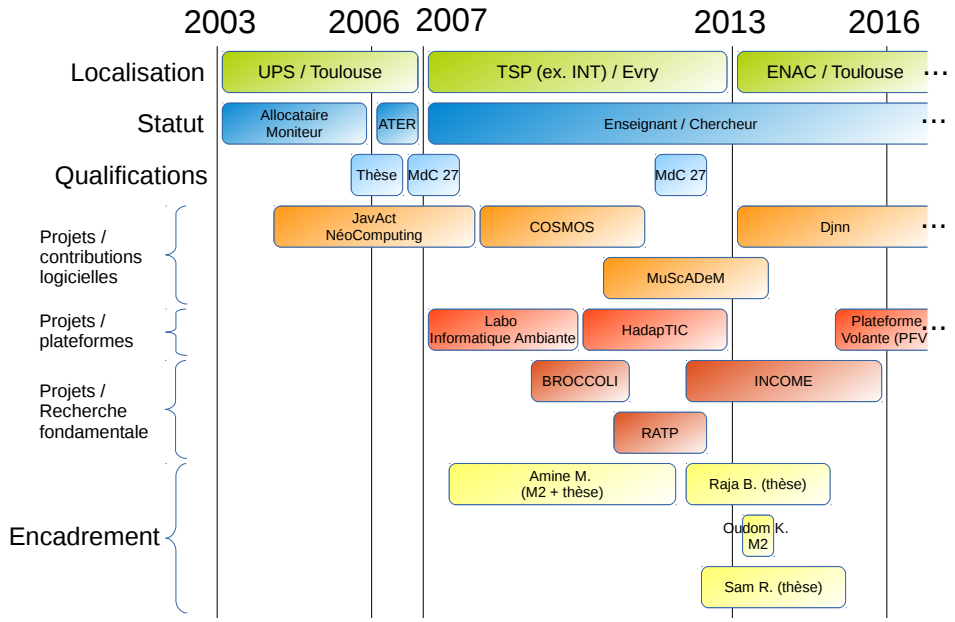

Figure 1.2 – Activités scientifiques

Sur le plan statutaire, à l'exclusion de mon expérience d'ATER à Toulouse, j'ai exercé dans des établissements qui ne dépendent pas du ministère de l'enseignement supérieur et de la recherche. Toutefois, chacun des établissements dispose d'une procédure interne similaire à la qualification, qui permet le recrutement d'un docteur sur un poste d'enseignant chercheur, avec un contrat de droit public. Cette procédure est facilitée néanmoins par l'obtention de cette qualification du CNU (section 27 en ce qui me concerne), c'est pour cela que j'ai choisi de refaire une qualification de MdC en 2013. Télécom Sud Paris est actuellement placé sous la tutelle conjointe du ministre chargé de l'industrie et du ministre chargé des communications électroniques. L'ÉNAC est sous tutelle du ministre chargé de l'aviation civile. Mon contrat actuel est à durée déterminée (il a été reconduit une première fois pour trois ans cette année), et pourra être reconduit en contrat à durée indéterminée en 2019. Mes activités d'enseignement sont précisées en annexe D.

Durant les dix dernières années, j'ai participé à des projets de recherche fondamentale, à des projets logiciels (principalement de niveau middleware) ainsi qu'à des projets de plateforme permettant de démontrer certains résultats ou de renforcer le réalisme de certaines expériences. En particulier, j'ai mis en place un laboratoire d'informatique ambiante à Télécom Sud Paris et contribué très fortement au montage de la plateforme HadaptTIC [2]. Je développerai les activités liées à la plateforme volante dans la partie prospectives/projet de recherche, projet que je dirige depuis cette année. Les publications liées à ces activités sont données en annexe B.

J'ai eu l'occasion de co-encadrer trois doctorants, Amine, Raja et Sam, qui ont tous soutenu leur thèse. En particulier, le financement de la thèse d'Amine a été obtenu sur une bourse "Institut Carnot", sur un projet scientifique que j'ai préparé et présenté et qui a été

2. http://hadaptic.telecom-sudparis.eu

intégré aux tâches menées dans le projet BROCCOLI (ARC INRIA). Ceux de Raja et Sam sont liés au montage du projet INCOME (ANR) qui a regroupé mon ancienne équipe de recherche toulousaine avec mon équipe évryenne de l'époque, mixant les thèmes recherches et aboutissant à une intéressante complémentarité. Des détails sont donnés en annexe C.

1.4 Plan

La partie qui suit contient un résumé des activités de recherche que j'ai menées depuis ma thèse. Elle est découpée en trois thèmes qui sont historiquement ceux sur lesquels je me suis penchés : l'adaptation des systèmes logiciels, leur déploiement, puis leur caractérisation multiéchelle. Une proposition de projet de recherche original conclue ce mémoire. En annexe, le lecteur trouvera mon CV détaillé, ma liste de publications, ainsi qu'une synthèse de mes activités d'encadrement et d'enseignement.

2 Activités de recherche antérieures

2.1 Des micro-architectures d'agents aux macro-architectures de systèmes

La démocratisation des moyens informatiques permet à la plupart des particuliers de disposer maintenant d'un ordinateur et d'un accès à un réseau. Au sein d'un foyer comme d'une entreprise, de nombreux produits auparavant électroniques deviennent informatisés. Pour le constater il suffit de considérer l'évolution des objets qui nous entourent, du réfrigérateur au modem en passant par l'aspirateur devenu robot, ou tant d'autres éléments de domotique, sans oublier les éléments nomades tels les téléphones portables, les ordinateurs portables (PDA, micro-portable) et peut-être bientôt d'autres formes de micro-périphériques embarqués. De même, la couverture réseau (en particulier sans fil) ne cesse de croître (campus universitaires, entreprises, lieux publics...) permettant une connectivité importante de ces périphériques. Certains auteurs parlent ainsi d'un univers d'informatique ambiante (*ubiquitous computing*, [Mattern, 2001]) dont la taille est gigantesque puisque l'Internet permet l'interconnexion de tous ces éléments.

Le réseau à *grande échelle* ainsi obtenu permet de disposer d'une quantité d'information extrêmement importante et d'offrir un grand nombre de services, dont la plupart sont encore à inventer. Le concept de grille de calcul [Foster and Kesselman, 1998] ou de stockage assimile celui-ci à un réseau de distribution semblable à la distribution d'eau ou d'électricité : de nombreux fournisseurs interconnectés offrent des services ou des données de manière plus ou moins standardisée à l'ensemble des utilisateurs accédant au réseau. On peut alors considérer ce réseau comme un système d'information très riche mais peu structuré. Les services les plus utilisés sont d'ailleurs les services de recherche d'information tel Google.

Ce réseau possède une dynamique d'utilisation très fluctuante et imprévisible, impliquant une très forte variation de la qualité de service. Son utilisation n'est pas du tout régulière mais connaît des pics de sur-utilisation et de sous-utilisation qu'il est parfois difficile de prévoir, de compenser ou d'exploiter [1]. Un site peut à tout instant, sans contrainte, décider de quitter le réseau ou de retirer certaines ressources ou certains services. Par ailleurs, le nombre d'éléments intervenant dans son architecture, même avec des taux de pannes extrêmement faibles et une redondance élevée, fait qu'en permanence de nombreux éléments matériels ou logiciels sont en panne.

1. Voir par exemple l'évolution en temps réel et les graphes d'historiques du trafic internet : http://www.internettrafficreport.com

Ces systèmes sont également très hétérogènes, depuis leurs composants matériels jusqu'au système d'exploitation en passant par leur connectivité. De plus, les systèmes étant administrés individuellement (par leurs propriétaires respectifs), ils peuvent évoluer de manière imprévisible et sans notification : par exemple mise à jour logicielle, passage d'une connexion X10 à une connexion sans fil type WiFi...

Le problème qui nous intéresse ici est celui du développement et de la maintenance des applications dans un tel contexte. Notre travail, de nature *génie logiciel*, a pour objectif de proposer des techniques et des outils (modèles, architectures, *middlewares*, *frameworks*...) qui contribuent à limiter la complexité induite par la prise en compte des problèmes (communs) posés par la grille et l'informatique ambiante : hétérogénéité, volatilité des ressources, volume des données échangées...

2.1.1 Exemples d'applications

Dans cette section, nous décrivons quelques exemples pour donner une vision plus concrète du contexte et des applications que nous envisageons, allant de l'informatique ambiante jusqu'au niveau grille. Les scénarios ci-dessous seront analysés dans la section suivante, et certains seront repris dans d'autres chapitres.

2.1.1.1 Une application de mutualisation de données

Présentation Considérons une application qui doit permettre à une communauté d'utilisateurs de fournir, de rechercher et d'exploiter des données réparties sur un réseau hétérogène de grande taille, par exemple le réseau mondial (du type application *pair à pair*). Chaque utilisateur connaît d'autres membres de la communauté et dispose de certaines connaissances sur les données qu'ils fournissent. Un utilisateur peut à la fois rendre accessible des données dont il est propriétaire, ainsi que rechercher puis utiliser des données parmi celles mises à disposition par la communauté. La recherche s'appuie sur les connaissances de l'utilisateur, et ces connaissances évoluent dans le temps en fonction des résultats obtenus.

Scénario d'utilisation Dans ce scénario, on considère les ressources (et pas simplement des données) sous forme de fichiers de type quelconque (son, image, texte...) ou bien de services (*Web service*, base de données...).

Soient cinq sites S_i d'une communauté, proposant chacun des ressources R_j, sauf S_4. Les liens de connaissances initiaux sont représentés sur la figure 2.1. Un utilisateur sur le site S_1 souhaite accéder à une ressource R_c. Celle-ci a été exploitée antérieurement sur S_2, donc la première tentative de recherche se fait sur ce site. Mais la ressource (notée R'_c) a évolué entre temps (mise à jour, suppression...) et s'avère inexploitable. Le système doit alors poursuivre la recherche sur les autres sites connus. S_4 ne répond pas (panne, déconnexion ou saturation...). Le site S_3 ne contient pas la ressource recherchée, mais connaît deux sites initialement inconnus de S_1. La ressource y est alors recherchée et elle est localisée puis exploitée sur S_5. Sur S_1, le logiciel ajoute le lien de connaissance vers S_5. L'exploitation de Rc peut être par exemple un téléchargement de fichier de S_5 vers S_1, ou bien l'exécution d'une requête sur une base de données du site S_5.

Il est possible d'aller plus loin. Par exemple lorsqu'un utilisateur cherche un composant logiciel, il souhaite probablement récupérer en plus de ce composant l'ensemble

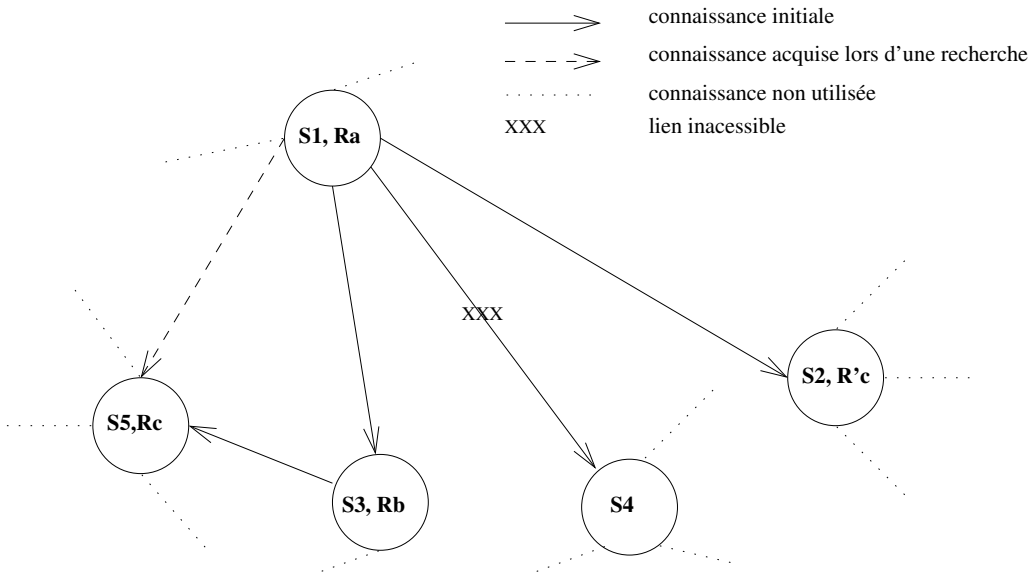

Figure 2.1 – Scénario d'utilisation du logiciel de mutualisation de ressources

de ses dépendances (composants requis). Dans ce cas, la recherche doit se poursuivre récursivement de manière automatique, jusqu'à ce que toutes les dépendances soient satisfaites.

Analyse L'application doit supporter les pannes et les déconnexions de certains sites ainsi que les évolutions des ressources (nouvelles, mises à jour, supprimées). Elle doit être capable de découvrir en cours d'exécution de nouvelles ressources et sites, et suffisamment flexible pour permettre l'utilisation de différents protocoles adaptés à chaque type de recherche, éventuellement fournis par le client.

2.1.1.2 Services dynamiques pour les véhicules

Présentation Avec la démocratisation des systèmes de navigation embarqués (GPS et bientôt sa version européenne Galileo) et l'augmentation des capacités des petits périphériques portables (PDA ou même téléphone portable), de nouvelles applications sont en train d'émerger. Dans un futur proche, il est envisageable que ces systèmes soient connectés au réseau Internet (type GPRS) et interconnectés de manière ad-hoc entre des véhicules proches (liaisons sans fil par exemple). Quelques projets s'intéressent à ce genre de scénarios, par exemple *Urban Vehicular Grid*[2], mais souvent en se focalisant sur des problématiques de niveau réseau. Il existe même un projet de norme WiFi, le 802.11p dit WAVE (Wireless Access for the Vehicular Environment), dédié à cette problématique.

Scénario d'utilisation La figure 2.2 montre un exemple d'interconnexion entre quatre véhicules V_i équipés des périphériques embarqués adéquats. A partir des informations de localisation fournies par son GPS (position, vitesse, direction), V_1 peut utiliser des services web de routage dynamique, ou encore d'autres services de géolocalisation (recherche

2. http://www.cs.ucla.edu/ST

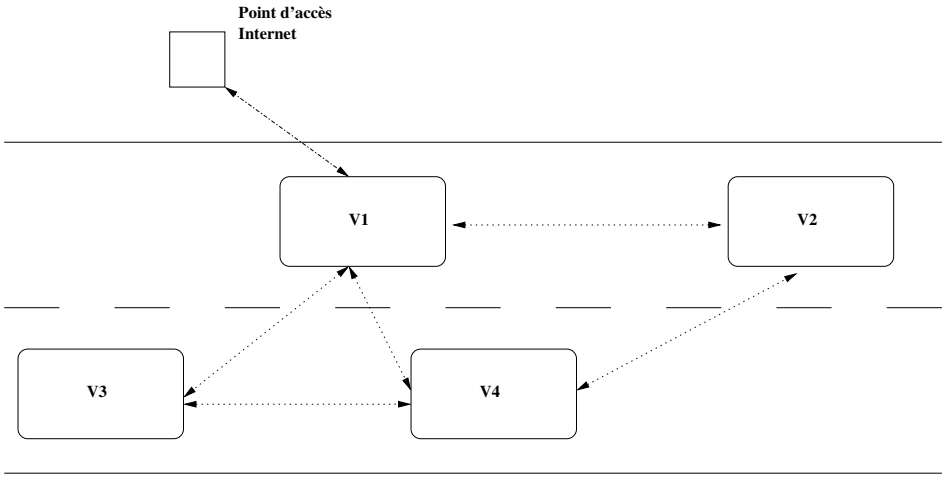

Figure 2.2 – Services dynamiques pour les véhicules

d'hôtel à proximité...). Etant connecté à des véhicules qui se déplacent en sens inverse (V_3 et V_4), il peut obtenir des informations sur les conditions de circulation à venir. S'il freine brutalement ou dévie brusquement de sa trajectoire, par exemple pour éviter un obstacle sur la voie, les véhicules qui suivent (ici V_2) peuvent être immédiatement prévenus et recevoir des alertes. De plus V_2, qui aimerait bien doubler V_1, peut connaître le nombre de véhicules à venir en face, ainsi que leurs paramètres (distance, vitesse...). Pour plus de sûreté, les codes réalisant ces opérations sont fournis et maintenus à jour (directement ou non) par les constructeurs de chaque véhicule.

Analyse En supposant l'existence de couches réseau adéquates (ce qui n'est pas, *a priori*, évident aujourd'hui), le fonctionnement du logiciel repose sur la possibilité de déployer du code fourni par des tiers (les constructeurs) et sur l'exploitation conjointe de plusieurs sources d'informations qui évoluent rapidement. La topologie des connexions évolue également rapidement (la portée d'un réseau WiFi [3] en extérieur est d'environ 400m, ce qui laisse un délai de 5 secondes environ sur une autoroute pour des voitures qui se croisent, soit une capacité de transmission d'environ 1Mo) et de manière imprévisible.

2.1.1.3 Distribution de modules logiciels

Dans une application de distribution (au sens de la livraison) de modules logiciels (applications entières ou parties d'application à des fins de mise à jour) sur un réseau [4], des producteurs de logiciel proposent les codes et leur documentation à des utilisateurs. Producteurs et utilisateurs interagissent par l'intermédiaire de distributeurs suivant deux modes distincts, le flot de données étant orienté des producteurs vers les utilisateurs. En mode *push*, le distributeur (éventuellement, en amont, le producteur) est l'initiateur de l'interaction avec l'utilisateur. Par exemple, il peut s'agir d'une mise à jour générale de tous les postes clients au sein d'une entreprise ou d'un groupe d'utilisateurs. Le distributeur doit d'abord identifier les utilisateurs cibles (éventuellement abonnés au service de distribution), ce qui peut demander une recherche sur le réseau et une analyse de l'existant chez les utilisateurs.

3. Avec la technologie 802.11b/g.

4. Cette activité est maintenant disponible dans de nombreux systèmes d'exploitation grand public (distribution Linux Debian, logiciels Microsoft, plugins Adobe Acrobat, Mozilla, Eclipse...).

En mode *pull*, c'est l'utilisateur (ou son système) qui est à l'origine de l'interaction et qui demande l'installation d'un module ; dans ce cas, c'est l'utilisateur qui doit trouver un distributeur et pouvoir consulter un catalogue qui décrit les modules disponibles.

Analyse La mise en œuvre de ces deux derniers exemples recouvre les problèmes de partage, de recherche et d'exploitation de ressources réparties mis en évidence plus haut. De fortes contraintes de qualité de service et de sécurité (confidentialité, intégrité) s'y ajoutent. Ces dernières conduisent, en particulier, à l'utilisation de techniques d'authentification et de chiffrement. En outre, dans le cas du DMP, le volume des dossiers influe également sur leur accessibilité et sur la performance du système.

Autres exemples Les possibilités d'applications ne s'arrêtent pas aux exemples précédents, nous en avons proposé d'autres (voir par exemple dans [Arcangeli et al., 2006]).

2.1.2 Problématique et motivations

Cette section reprend les analyses faites pour les exemples proposés, en étendant les constatations au cadre général des systèmes distribués à grande échelle.

2.1.2.1 Facteur d'échelle et contexte ouvert

Les applications évoquées s'articulent autour de différentes unités physiques ou logiques (PDA, cluster, PC portable… et serveur d'application, suite logicielle, composant logiciel…) qui fournissent ou exploitent des ressources ou des services. Ces unités sont autonomes, réparties (leur exécution peut donc être parallèle), et **administrées indépendamment** les unes des autres. Le contrôle est **décentralisé** et distribué à travers ces unités.

La principale caractéristique de ces applications est leur **échelle**. Ici, la notion d'échelle recouvre à la fois le nombre d'unités et les dimensions du réseau : le nombre d'unités peut être très grand et leur couplage faible (interconnexions via un réseau filaire de grande taille ou sans fil, à faible qualité de service). Dans ces conditions, le nombre et les volumes des interactions entre les unités influent fortement sur le fonctionnement et la performance.

Une autre caractéristique, conséquence de l'échelle, est l'**ouverture** des systèmes : outre les problèmes de pannes, les unités peuvent intégrer ou quitter le système dynamiquement, voire se déplacer sur le réseau, sans contrôle global et sans qu'il ne soit possible de prévoir l'ensemble des évolutions. De l'ouverture, résultent les problèmes de **disponibilité** (ou volatilité) des ressources et des services. De plus, les utilisateurs peuvent ne disposer que d'une connaissance partielle ou ancienne de sa structure ainsi que des informations sur les services disponibles.

Ces applications se distinguent également par la nature **hétérogène** (matériel, système d'exploitation, connexion réseau) des systèmes à exploiter. De fait, pour les faire interopérer, il est nécessaire d'utiliser des protocoles communs, le plus simple étant d'exploiter des standards pour en faire des systèmes ouverts.

2.1.2.2 Besoins de flexibilité

Pour être aussi efficaces que possible, les applications réparties à grande échelle doivent supporter différents types d'évolutions [Boinot, 2002]. Elles doivent être *configurables*, c'est-à-dire capables d'adaptation statique (réalisée en dehors de leur utilisation) pour faciliter leur maintenance. Elles doivent être *extensibles*, pour permettre l'intégration de nouvelles fonctionnalités correspondant à des besoins identifiés après la livraison. Enfin elles doivent être *dynamiquement adaptables*, pour se reconfigurer en cours d'exécution afin de présenter un mode de fonctionnement optimal. Cela peut aller de la simple variation de paramètre à la reconfiguration complète de l'architecture logicielle en cours de fonctionnement. Nous parlons de *flexibilité* lorsqu'un système ou une application présente ces trois capacités d'évolution [5].

Adaptation statique Dans tout projet logiciel, les activités de maintenances sont nécessaires pour conserver un système opérationnel et satisfaisant pour les utilisateurs. On distingue plusieurs catégories de maintenance : corrective (correction des défauts résiduels du logiciel), adaptative (suite à une modification d'une spécification), perfective (amélioration du fonctionnement), évolutive (ajout de fonctionnalité) ou préventive (enrichissement du logiciel par des mécanismes de traitement d'erreurs). Les besoins d'adaptation statiques doivent être pris en compte dès la conception, en proposant des architectures logicielles adéquates, permettant leur configuration.

Adaptation dynamique Lors de la construction ou du déploiement de systèmes ouverts répartis à grande échelle, on ne peut pas faire d'hypothèse forte sur la disponibilité ou la forme d'un service [6], ni sur l'organisation globale du système. La robustesse est cependant une exigence forte. Les applications doivent supporter les pannes et les déconnexions de certains sites ainsi que les évolutions des ressources (ajout, mise à jour, suppression). Elles doivent être suffisamment souples pour répondre à différents besoins en terme de localisation et d'exploitation des ressources et s'adapter (dynamiquement) à la disponibilité et à la qualité des services (y compris celle du réseau de communication).

Les techniques d'adaptation dynamique visent à prendre en compte « instantanément » les variations du contexte d'exécution, de façon non seulement à continuer à faire fonctionner l'application mais aussi à tirer le meilleur parti de la nouvelle configuration. Toutefois, l'intégration de mécanismes d'adaptation dynamique peut se révéler complexe, et il est donc nécessaire de disposer d'outils pertinents pour réduire cette complexité. De manière similaire à l'adaptation statique, la prise en compte de ces besoins doit être faite dès la conception, en proposant des architectures logicielles adéquates, permettant leur reconfiguration dynamique.

Personnalisation Les services génériques ou les ressources mises à disposition sur le réseau peuvent ne pas être directement exploitables par un client. Soit par un manque au niveau logiciel (application, codec…), soit par des capacités de traitement (bande passante, périphériques, etc.) inappropriées. Ainsi, certains de ces exemples requièrent des capacités de personnalisation pour adapter un service aux besoins du client.

5. Il ne semble pas y avoir dans la littérature de terminologie formelle ; aussi nous proposons d'employer le terme *flexibilité* pour regrouper toutes les capacités d'adaptation d'un système citées dans ce paragraphe.

6. On emploie ici assez indifféremment les termes « service » et « ressource » (voire « information »).

Pour cela, une première approche consiste à faire varier des paramètres du côté du système qui rend le service. En allant plus loin, on peut permettre au client de transmettre non plus un ensemble de paramètres mais un programme pour adapter le service en fonction de ses besoins. Dans le cas de la recherche d'information, on peut ainsi permettre à l'utilisateur de déployer son propre algorithme de recherche, pour augmenter la pertinence des résultats trouvés.

2.1.2.3 Implications sur la conception

La conception de programmes concurrents répartis est beaucoup plus complexe et source d'erreurs que celle des programmes séquentiels centralisés. Ces difficultés sont liées à la multiplication d'entités (processus…) ainsi qu'aux problèmes de communication et de synchronisation. Une réponse possible consiste à utiliser des approches de plus haut niveau dans toutes les phases de conception.

Outre les aspects fonctionnels, les principaux besoins à prendre en compte lors de la conception sont les suivants :

Recherche de ressources Il faut intégrer aux applications la capacité de rechercher dynamiquement des services (au sens de la localisation sur le réseau, et de la découverte de nouvelles ressources), puis de les spécialiser afin de les adapter aux conditions d'exécution et aux besoins. Par exemple, la localisation sur le réseau de ressources et de services peut être plus efficace en utilisant une algorithmique spécifique personnalisée qui dépend des connaissances et des besoins du demandeur.

Organisation des traitements répartis Ce besoin concerne l'exploitation à distance ou locale, la gestion des volumes et des flux et le déploiement de services : il s'agit de prendre en compte le coût des échanges sur le réseau et d'adapter l'organisation à l'état du système ou aux spécificités de l'application (un compromis lié au coût de transfert des données et de déploiement de service est nécessaire au niveau du choix du lieu de traitement). Par exemple, déporter sur un site serveur un traitement spécifique qui met en œuvre l'expertise d'un client peut permettre de réduire le nombre et le volume des données échangées sur le réseau qui sont nécessaires à l'interaction ou qui en résultent.

Pour déporter et exécuter les traitements répartis, il est nécessaire de les déployer. Le déploiement de logiciel [Carzaniga et al., 1998c] recouvre différentes activités postérieures au développement dont l'objectif est de rendre un logiciel opérationnel pour un utilisateur : transfert du producteur à l'utilisateur, configuration (adaptation au contexte d'utilisation), installation et activation de composants logiciels, mise à jour et reconfiguration (évolution) pour les logiciels en production. La complexité et l'importance du déploiement ont augmenté récemment avec l'évolution des réseaux et la construction d'applications à base de composants. Il en résulte un besoin d'automatisation de cette phase du cycle de vie du logiciel.

Sûreté de fonctionnement Cette préoccupation concerne l'amélioration de la qualité du logiciel en le rendant globalement plus fiable ; les principaux mécanismes sont la tolérance aux pannes, la gestion du mode déconnecté, ou le support de l'hétérogénéité.

Sécurité Il s'agit de prendre en compte les potentielles attaques du logiciel, et de proposer des parades ou des moyens de prévention ; par exemple, en prévoyant des services d'authentification des entités et de leurs représentants, ou encore des communications sécurisées...

2.1.3 Micro-architecture : modèle d'agent mobile adaptable

Les applications décrites précédemment, et plus généralement les systèmes déployés dans des environnements de répartition à grande échelle, doivent faire face à l'hétérogénéité des données, des logiciels et du matériel, à la volatilité potentielle des services ainsi qu'à de fortes variations (qualitatives et quantitatives) des conditions d'exécution. Ces différents problèmes doivent être pris en compte lors du développement et les applications doivent être capables de s'adapter dynamiquement.

Les intergiciels traditionnels permettent aux développeurs de s'abstraire de certains aspects techniques, souvent délicats, liés à la répartition et aux opérations distantes. Les intergiciels adaptables ont pour objectif supplémentaire de fournir des modèles et des services pour la construction et la maintenance d'applications robustes et dynamiquement adaptables aux conditions d'exécution et de contribuer ainsi à la maîtrise de la complexité du développement. Notre démarche s'inscrit dans cette approche : nous avons étudié et développé un intergiciel générique à base d'agents mobiles adaptables par remplacement et spécialisation de composants.

2.1.3.1 Besoins d'adaptation individuelle des agents

La mobilité d'agent permet de rapprocher client et serveur et en conséquence de réduire le nombre et le volume des interactions distantes (en les remplaçant par des interactions locales), de spécialiser des serveurs distants ou de déporter la charge de calcul d'un site à un autre. Une application construite à base d'agents mobiles peut se redéployer dynamiquement suivant un plan pré-établi ou en réaction à une situation particulière, afin par exemple d'améliorer la performance ou de satisfaire la tolérance aux pannes, de réduire le trafic sur le réseau, ou de suivre un composant matériel mobile.

Ainsi, la mobilité est un mécanisme important d'adaptation (complémentaire) et les agents mobiles sont un outil à part entière pour l'adaptation des systèmes. Toutefois, l'adaptation au niveau de l'application n'est pas suffisante, et nous discutons dans cette section des besoins d'adaptation statique des agents (étapes de modification d'un système en dehors de son exécution), puis des besoins d'adaptation dynamique, lors de l'exécution des agents mobiles et particulièrement lors de leurs déplacements. Ce niveau d'adaptation (intra-agent) vient compléter les différentes possibilités d'adaptation des systèmes.

Adaptation statique Dans le coût total d'exploitation d'un logiciel, les parts liées à la maintenance sont plus importantes que celles liées au développement. Pour les mainteneurs, il s'agit de pouvoir ajouter de nouvelles fonctionnalités, corriger des fautes de conception ou adapter le système à d'autres plates-formes par exemple. Pour les utilisateurs, il s'agit de choisir une configuration adaptée à ses besoins, identifiés *a priori*, par exemple parmi différentes interfaces (IHM) ou protocoles de communication.

Les besoins d'adaptation statique se retrouvent à toutes les échelles du logiciel, et en

particulier au niveau des architectures que nous proposons. En fournissant une architecture souple et capable d'évolution, la maintenance est facilitée et par conséquence, le coût total est réduit. Ainsi, les agents doivent pouvoir être configurés à froid, c'est-à-dire avant leur exécution, et offrir des mécanismes d'évolution de leur architecture.

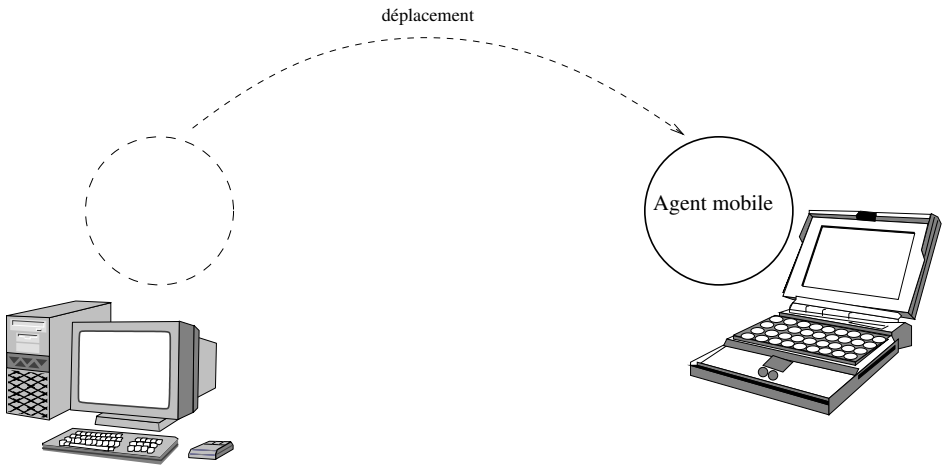

Figure 2.3 – Besoins d'adaptation dynamique pour les agents mobiles

Adaptation dynamique au contexte d'exécution Prenons l'exemple (fig. 2.3) d'un agent mobile créé sur un ordinateur relié à un réseau local filaire isolé de l'extérieur par des protections adéquates (pare-feu...) qui se déplace ensuite vers un environnement non fiable du point de vue de la sécurité des communications et avec des pertes potentielles du signal réseau (typiquement un portable dans un environnement public relié au réseau par liaison sans fil de type WiFi). Cet agent peut utiliser à l'origine des protocoles de communications ayant des propriétés non-fonctionnelles de faible sûreté, faible sécurité et forte performance. Après la mobilité, il serait judicieux grâce à des mécanismes d'adaptation dynamique d'utiliser des protocoles dont la sémantique est identique, mais ayant des propriétés non-fonctionnelles de forte sûreté et de forte sécurité, éventuellement au détriment de la performance.

Dans d'autres cas, il peut être nécessaire d'adapter le protocole de localisation des agents mobiles, les protocoles de communication (suivant le type de connexion), les informations et les services mis à disposition sur les sites visités (découverte et accès à des périphériques, bases de données, informations géographiques...).

La mobilité n'est pas la seule cause des besoins d'adaptation dynamique. En effet, l'environnement d'un agent immobile peut changer, soit dans le cas où le système physique est lui-même mobile (et où l'on retrouve les mêmes besoins que ceux cités plus haut), soit dans les cas où l'environnement peut changer (évolution de la connectivité, des logiciels ou encore des périphériques connectés).

Ainsi, il est nécessaire de permettre l'adaptation dynamique des mécanismes d'exécution non fonctionnels des agents mobiles aux conditions d'exécution qu'ils rencontrent durant leur vie, pour offrir un maximum de services utiles et efficaces aux applications qui les utilisent.

Origine de l'adaptation Pour rester conforme au principe d'autonomie des agents, l'adaptation dynamique doit être pilotée et réalisée de manière interne exclusivement[7]. Ainsi, dans un intergiciel à base d'agents, les adaptations dynamiques devraient avoir deux origines :

— depuis l'application, par une instruction du niveau du langage de description du code fonctionnel pour répondre à un besoin applicatif particulier (bascule des communications en mode crypté...),
— depuis l'intergiciel, lorsque le système d'accueil détecte un changement dans l'environnement et ce via un organe de décision interne à l'agent. Toutefois, l'adaptation déclenchée par un tel changement n'est qu'une notification qu'il peut ignorer. Ainsi le contrôle de l'adaptation est entièrement à la charge de l'agent.

2.1.3.2 Principes architecturaux

Pour réaliser les solutions répondant aux besoins d'adaptation que nous avons montrés, nous proposons d'aborder le problème en proposant une organisation architecturale adéquate (configurable et reconfigurable) d'un agent mobile. Pour mettre en œuvre une telle architecture, nous utilisons :

— d'une part les principes de séparation des préoccupations et de modularité,
— d'autre part les technologies à base de composants pour réutiliser des éléments existants, permettre leur configuration et apporter les mécanismes d'adaptation dynamique.

De plus, le niveau d'adaptation assez fin que nous proposons nécessite de réaliser de la ¡¡ micro-chirurgie ¿¿ sur l'architecture des agents, ce qui implique l'emploi d'unités de structuration à grain fin pour les éléments adaptables (mécanismes de communication, de mobilité...).

Principes généraux Les principes généraux suivants devront être pris en compte :

— *Transparence* : l'adaptation des agents est une préoccupation transversale, elle doit donc s'effectuer de manière transparente du point de vue de l'application agent.
— *Adaptation individuelle* : pour conserver le principe d'autonomie, chaque agent doit posséder ses propres capacités d'évolution, indépendamment des choix de configuration des autres agents du système.
— *Séparation des préoccupations* : nous nous intéressons à l'adaptation dynamique des propriétés non-fonctionnelles des agents, celles-ci doivent être séparées de manière physiques, afin d'être séparément remplaçables.
— *Faible granularité* : l'isolation physique des mécanismes adaptables et la finesse de l'adaptation souhaitée imposent l'emploi d'éléments de faible granularité (échelle d'un service unique).

Architecture à méta-objets Les protocoles à méta-objets permettent de répondre aux besoins de séparation des préoccupations et de transparence des mécanismes d'adaptation et des services adaptables. Au niveau de base, le code fonctionnel dispose de primitives particulières, réifiées au méta-niveau sous forme de méta-objets. Il devient possible de manipuler

7. Nous employons le terme d'*agent adaptable* là où certains utilisent plutôt *auto-adaptatif* pour caractériser ces propriétés d'adaptation autonome, et d'autres proposent le terme *adaptatif*.

ces méta-objets et en particulier de les modifier, pour adapter individuellement les services rendus au niveau de base, sans que celui-ci en soit informé.

L'architecture que nous proposons est inspirée de l'architecture réflexive proposée dans [Marcoux et al., 1998] (elle-même dérivée de CodA [McAffer, 1995]) et reprise dans JAVACT 4. Le niveau de base contient le code fonctionnel de l'agent (comportement déterminé par le programmeur de l'application agent) et le niveau méta décrit les mécanismes opératoires de l'agent.

Micro-composants remplaçables dynamiquement Pour spécialiser le méta-niveau d'un agent et le doter de nouveaux mécanismes opératoires, une première approche consisterait à procéder par héritage. Mais, dans ce cas, il serait impossible d'adapter dynamiquement les mécanismes internes des agents. L'approche par délégation, bien plus flexible, s'impose. Le paradigme composant (modularité et mécanismes d'assemblage) nous semble un bon candidat pour supporter le codage des différentes composantes des agents identifiées plus haut. Les agents sont alors construits par assemblage de composants (ce sont donc des composites), les composants fonctionnels étant placés au niveau de base, tandis que les composants non-fonctionnels sont regroupés dans le méta-niveau.

Pour une identification précise des rôles et donc pour simplifier la validation des assemblages, nous proposons l'utilisation de composants de granularité minimale c'est-à-dire n'offrant qu'un service opératoire unique. Ainsi, nous proposons d'employer le terme *micro-composants*, pour représenter les méta-composants à grain fin qui correspondent aux services non-fonctionnels du méta-niveau.

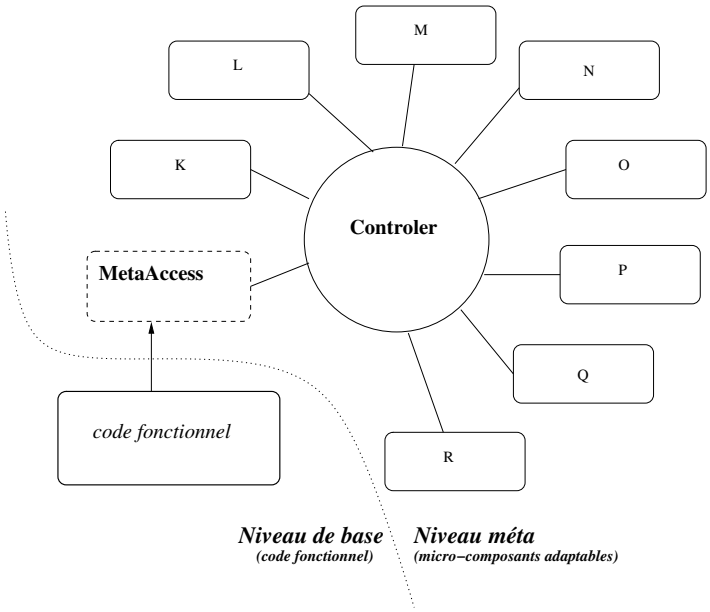

Figure 2.4 – Architecture en étoile autour d'un connecteur

Définition d'un connecteur Pour permettre l'adaptation dynamique des micro-composants, nous proposons d'utiliser une architecture en étoile [Leriche and Arcangeli, 2004], dans laquelle les micro-composants sont reliées à un connec-

teur nommé *contrôleur* (`Controler` sur la figure 2.4). Ce connecteur permet d'implémenter le principe de délégation pour faciliter l'évolution de l'architecture. Il est le point d'entrée unique sur les services offerts par les micro-composants et contient également un ensemble de primitives pour permettre le changement des micro-composants par substitution.

L'encapsulation par le contrôleur des appels de méthodes d'un composant autorise l'utilisation d'un composant standard par défaut, autant que d'un composant spécialisé par héritage ou utilisant par délégation les services d'un ou plusieurs autres composants. On peut ainsi envisager l'utilisation de micro-composants composites dans cette architecture.

Interface entre les niveaux Un objet particulier (`MetaAccess`) permet au code du comportement situé au niveau de base d'accéder au méta-niveau. Cela permet une restriction d'accès aux seules primitives autorisées au niveau de base, à l'exclusion des manipulations du niveau supérieur (changement explicite de configuration, récupération des références des composants, etc.). Cette séparation contribue à la sûreté et à la sécurité de l'architecture d'agent adaptable.

Système d'accueil Le rôle d'un système d'accueil est d'offrir des mécanismes de création et d'hébergement local d'agents. Afin de pouvoir être informé des variations de l'environnement, il englobe un ensemble de sondes de bas niveau (du type décrit dans [da Silva e Silva et al., 2003] par exemple). Lorsqu'une des sondes détecte un changement, elle émet une notification au système d'accueil qui la répercute à l'ensemble de ses agents.

Les agents conservent leur autonomie, car ils peuvent décider d'ignorer les notifications d'évolution de l'environnement d'exécution. De plus, il n'y a pas de dépendance fonctionnelle entre les agents créés sur une place et son système d'accueil, celui-ci n'offrant que des ressources d'exécution.

La gestion de l'hétérogénéité doit être réalisée par le système d'accueil, généralement spécifique à la plate-forme physique qui l'héberge, et qui propose aux agents une vision uniforme des évènements et des informations sur le contexte d'exécution.

Analyseur Une politique d'adaptation est un ensemble de règles définies pour un ensemble de composants d'un agent et un ensemble d'évènements liés aux variations du contexte d'exécution. Un analyseur implémente une politique d'adaptation. Lorsqu'une sonde détecte un changement dans l'environnement, elle prévient le système d'accueil qui émet à son tour une notification individuelle à chaque agent vivant sur son site. Ceux-ci peuvent ignorer cette notification, sinon elle est prise en charge par l'analyseur. En fonction de sa politique d'adaptation, l'analyseur peut décider du remplacement d'un composant. Il utilise les services du contrôleur pour les effectuer.

Par exemple, soit un ensemble de micro-composants : K, K', L, M avec K et K' du même type (ils implémentent la même interface), donc qui sont interchangeables. On dispose de A^δ, un analyseur spécifique à cet ensemble de composants, contenant la règle suivante :
$A^\delta : p \rightarrow K \Leftrightarrow K'$
qui remplace le composant K par K' lorsqu'une propriété p de l'environnement est vraie.

Les agents sont initialement créés avec les micro-composants K, L, M et l'analyseur A^δ. La propriété p n'est pas vérifiée dans le contexte d'exécution.

Lorsque la sonde de l'écouteur d'environnement détecte le passage de $\neg p$ à p, le système

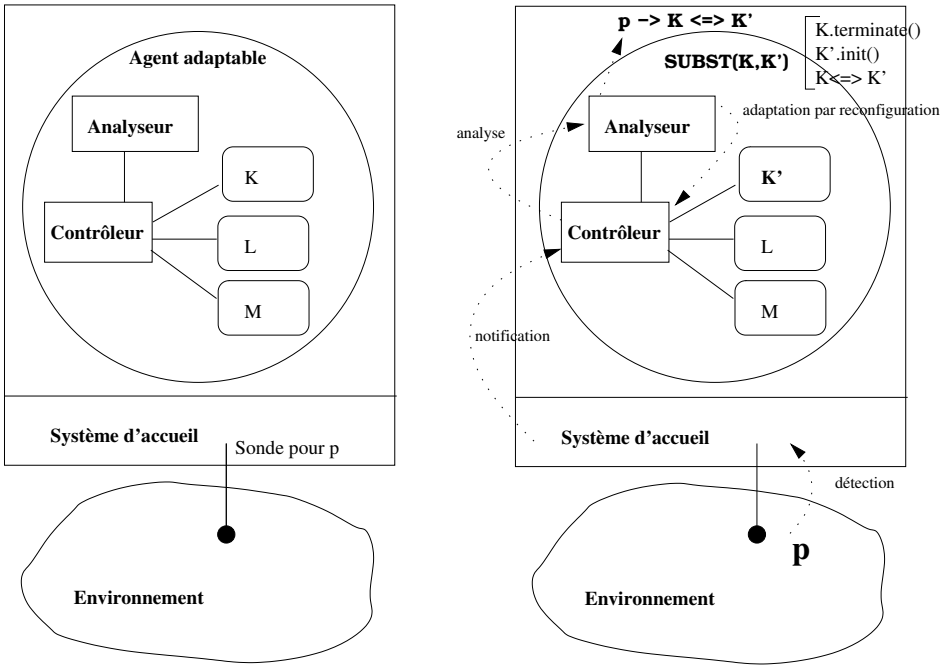

Figure 2.5 – Mécanisme de changement dynamique des micro-composants

d'accueil envoie une notification à chacun de ses agents. L'analyseur prend la main lors des phases d'attente bloquante (réception d'un message par exemple).

Celui-ci connaît l'état précédent de l'environnement, notamment $\neg p$. Il découvre que le changement consiste en un passage de $\neg p$ à p. Sa règle d'adaptation est donc vérifiée : $p \rightarrow K \Leftrightarrow K'$. Il utilise alors les services du contrôleur pour appliquer la partie opératoire de la règle, changer K en K' (*SUBST(K,K')* sur le schéma).

Au niveau du contrôleur, le remplacement d'un micro-composant se fait en trois étapes : terminaison du précédent, initialisation du prochain, puis enfin permutation. A l'issue de ces opérations, l'agent a été dynamiquement adapté. La cohérence de la configuration des micro-composants doit être gérée par le programmeur de l'analyseur et les changements peuvent être différés à une période d'inactivité de l'agent pour être sûr que l'adaptation ne génère pas de conflit avec son exécution (cf. section suivante pour les détails d'implémentation).

Conformément au principe de séparation des préoccupations, l'analyseur contient les règles d'adaptation aux variations du contexte d'exécution et le contrôleur sert uniquement d'opérateur mécanique pour réaliser l'échange dynamique des micro-composants.

2.1.3.3 Modèle d'agent mobile adaptable

Le concept d'*agent* logiciel est similaire à celui de composant, dans la mesure où il n'en existe pas de définition très précise. De ce fait, il existe de nombreux modèles et implémentations, basés sur des propriétés variables. L'architecture que nous proposons repose sur le modèle d'acteur [Agha, 1986], que nous justifions ci-dessous.

Choix du modèle d'acteur Les acteurs sont des entités anthropomorphes qui communiquent par messages. Au sein d'une communauté, un acteur est uniquement connu par sa référence. La communication est point à point, asynchrone, unilatérale et supposée sûre. Les messages reçus sont stockés dans une boîte aux lettres et traités en série (en exclusion mutuelle). Le comportement décrit la réaction de l'acteur à un message. Le comportement est privé : il contient les données et les fonctions de l'acteur et constitue son état. L'acteur encapsule données et fonctions mais également une activité (fil d'exécution) unique qui les manipule. Un comportement d'acteur n'est donc, à un instant donné, traversé que par un fil d'exécution au plus (la synchronisation est implicite).

Lors du traitement d'un message, un acteur peut créer (dynamiquement) de nouveaux acteurs, envoyer des messages aux acteurs qu'il connaît et changer de comportement c'est-à-dire définir son comportement pour le traitement du prochain message. Le changement de comportement est un mécanisme puissant d'évolution et d'adaptation individuelle. D'une part il permet à un acteur de modifier dynamiquement la nature des services qu'il fournit (évolution de la sémantique), d'autre part il permet de faire évoluer son interface. Dans ce dernier cas, il n'est pas possible de garantir dans tous les cas qu'un message envoyé pourra effectivement être traité par son destinataire. Cependant, pour le programmeur, le changement de comportement offre une alternative à la gestion de variables d'état et à l'utilisation de gardes, qui accroît l'expressivité. Programmer un acteur revient donc à programmer ses comportements, l'enchaînement des comportements étant décrit dans les comportements eux-mêmes.

L'idée de l'utilisation des acteurs pour la programmation d'applications concurrentes et réparties n'est pas neuve. Néanmoins, elle semble particulièrement pertinente dans le cadre du calcul réparti à grande échelle de par les propriétés qui distinguent les acteurs du modèle d'objet classique (encapsulation de l'activité, changement d'interface, communications asynchrones) [Arcangeli et al., 2001] :

— d'une part, la communication par messages asynchrones est bien adaptée aux réseaux de grande taille, voire sans fil, où les communications synchrones sont trop difficiles et coûteuses à établir,
— d'autre part, l'autonomie d'exécution et l'autonomie comportementale favorisent l'intégration de mécanismes de mobilité et garantissent un certain niveau d'intégrité.

La mobilité dans le modèle d'acteur La mobilité d'acteur peut être naturellement définie en se basant sur le traitement des messages en série et le changement de comportement, ainsi que sur la création dynamique et l'envoi de messages [Arcangeli et al., 2000]. Lors du changement de comportement (donc, entre deux messages), l'acteur se réduit d'une part au contenu de sa boîte aux lettres et d'autre part au comportement défini pour le traitement du prochain message. Par nature, le comportement est une entité de première classe qui matérialise l'état et qui est manipulable par programme. Il est alors possible de créer à distance un autre acteur défini à partir du comportement. On obtient ainsi simplement une évolution de l'acteur d'origine à qui on peut faire suivre tous les messages qui étaient en attente. L'état est donc entièrement transporté (via le comportement) et restauré sans que le programmeur n'ait à développer de code spécifique pour cela. Ainsi, après migration, l'acteur peut poursuivre son activité à distance en bénéficiant des acquis résultant des traitements de messages précédents. La mobilité est ainsi différée au moment où l'acteur change de comportement.

Tout acteur est donc potentiellement mobile et, ainsi définie, la mobilité n'altère pas la

sémantique des programmes. On peut noter que les problèmes de cohérence de copies et de synchronisation que l'on rencontre quand on introduit la mobilité dans le modèle objet ne se posent pas ici du fait de l'unicité du fil d'exécution et de l'exclusion mutuelle sur le traitement des messages.

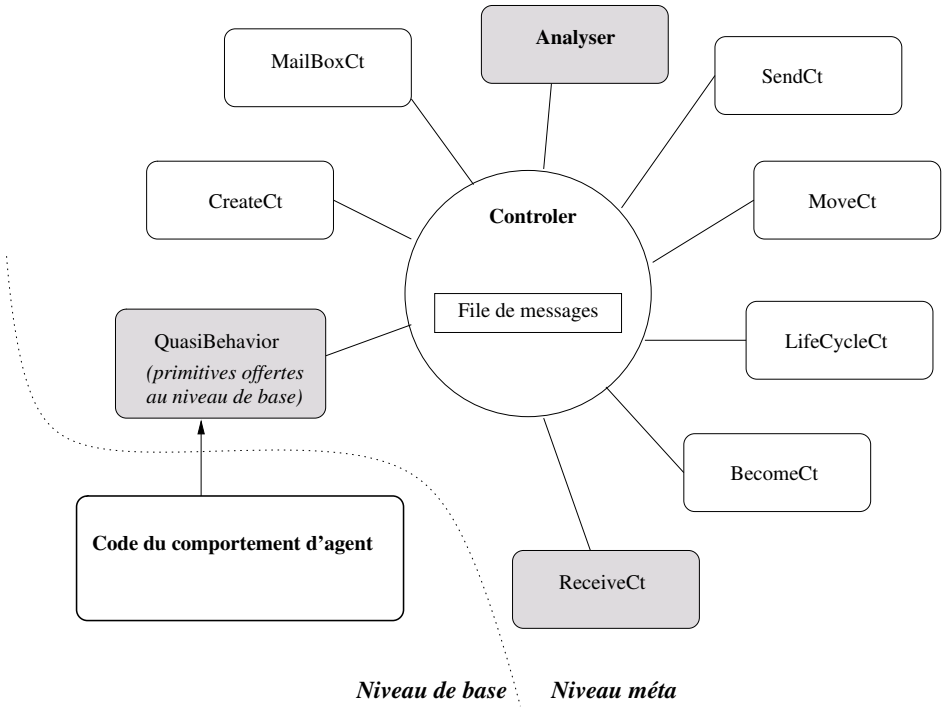

Figure 2.6 – Architecture d'agent mobile adaptable

Architecture d'agent mobile adaptable A partir du modèle d'acteur et en appliquant les principes proposés dans la section précédente, nous obtenons une architecture de micro-composants en étoile autour d'un connecteur (figure 2.6). Les différentes capacités des acteurs ont été réifiées sous la forme de micro-composants :

- `SendCt` pour l'envoi de message,
- `CreateCt` pour la création locale et distante d'agents,
- `BecomeCt` pour le changement de comportement,
- `MoveCt` pour la mobilité.

D'autres mécanismes opératoires internes à l'agent ont également été réifiés :

- `MailBoxCt` pour la gestion de la boîte aux lettres,
- `LifeCycleCt` pour activer et gérer le cycle de vie de l'agent.

Les micro-composants grisés sur le schéma (figure 2.6) n'ont pas été prévus pour être remplaçables. En effet, ils ne correspondent pas à un mécanisme que nous souhaiterions réifier.

Pour le micro-composant de réception de message, il s'agit d'un choix contraint par la conception. En effet, il constitue le point d'entrée d'activation d'un agent, et modifier ce micro-composant reviendrait à perdre la connaissance de ce point d'entrée, ce qui empêcherait toute communication entrante avec des agents qui possédaient la référence du micro-composant remplacé. Enfin, le `QuasiBehavior`, n'est pas réellement un micro-composant, il implémente le Meta-Access décrit plus haut. Il sert d'interface entre le niveau

de base et le méta-niveau, pour renforcer la sûreté et la sécurité de l'architecture. Son adaptation ne nous semble pas intéressante (pas de besoin *a priori*).

Il serait possible d'implémenter des fonctionnalités partagées, via des micro-composants communs à un ensemble d'agents ou à l'ensemble des agents d'un système d'accueil, en particulier pour factoriser les mécanismes de communication. Nous pensons que dans un objectif d'autonomie et de capacité d'adaptation dynamique autonome, les agents doivent être complètement indépendants. De plus, factoriser les micro-composants de communication (`SendCt` et `ReceiveCt`) au niveau du système d'accueil risquerait de provoquer un goulot d'étranglement (en particulier pour la réception) et donc une dégradation des performances globales d'un système d'agents.

2.1.4 Micro-architecture : modèle d'agent flexible

Les agents mobiles sont par nature, des outils d'adaptation en contexte réparti. Afin d'accroître les possibilités d'adaptation, nous avons proposé un niveau de flexibilité intra-agent contrôlé par l'agent lui-même et nous avons défini un modèle d'agent mobile configurable et auto-adaptable dynamiquement. Dans cette section, nous proposons un niveau supplémentaire d'adaptation et nous introduisons le modèle d'agent flexible.

2.1.4.1 Besoins d'adaptation supplémentaires

Le modèle d'architecture d'agent mobile que nous avons présenté précédemment permet la configuration des agents par un choix de micro-composants lors de leut création, ainsi que leur adaptation dynamique au contexte d'exécution par le remplacement de certains de ces micro-composants. Toutefois, la structure et le squelette de l'architecture de l'agent sont figés et les adaptations sont limités à un ensemble de micro-composants dépendant de ce choix d'architecture. Dans notre implémentation, il s'agit d'un ensemble de micro-composants réifiant les services non-fonctionnels nécessaires à l'exécution d'un acteur mobile.

Vers davantage de flexibilité Prenons l'exemple de notre prototype de logiciel pair à pair cité précédemment. Il est implanté par un système d'agents répartis. Certains agents sont des agents d'interface (avec les pairs clients ou serveurs). Certains agents sont mobiles sur le réseau et peuvent être amenés à percevoir leur environnement physique (charge du réseau, niveau de sécurité…) ; pour s'y adapter, ils peuvent redéfinir leurs mécanismes de communication ou de déplacement. Certains agents peuvent développer une algorithmique complexe (recherche de ressource, exploitation d'une ressource…). En termes d'évolution, de décision, de perception, de réactivité et de mobilité, les différents agents qui constituent le système ont des caractéristiques et des besoins variables. Or, dans notre implantation, tous ont du être développés à partir du modèle d'agent unique offert par le *middleware*.

Plus précisément, dans la proposition mise en œuvre dans le *middleware* JAVACT, l'adaptation se réduit au choix initial des micro-composants et au remplacement dynamique d'un micro-composant par un autre qui offre le même service (spécifié par une interface Java). L'évolution des services opératoires est limitée par une interface fixe (limitation de l'évolution au changement d'implémentation) et il n'est pas possible de décider quels micro-composants sont remplaçables et lesquels ne le sont pas. L'architecture de l'agent

est également figée : les agents sont des *acteurs* [Hewitt, 1977, Agha, 1986] répartis et mobiles et il n'est possible ni d'introduire un nouveau micro-composant ni de retirer un micro-composant inutile (par exemple, le gestionnaire de déplacement pour un agent immobile).

D'autre part, on peut aussi noter que notre expérience de la maintenance du *middleware* JAVACT nous a conduit à réviser plusieurs fois l'architecture des agents et à modifier le *middleware*. Par exemple :

— lors de l'introduction de la mobilité, il a fallu ajouter au méta-niveau un micro-composant dédié (il interagit avec d'autres composants de méta-niveau) et donc modifier l'architecture des agents ;
— pour mettre en œuvre un mécanisme de communication synchrone, nous avons dû modifier les interfaces existantes, en l'occurrence celles des micro-composants de communication pour éviter d'intégrer un micro-composant supplémentaire ;
— nos agents (modèle d'acteur) se sont révélés mal adaptés pour implanter des agents AMAS [Capera et al., 2003] (lors d'une collaboration avec une équipe du domaine des sytèmes multi-agent) et une nouvelle architecture d'agent a dû être définie [Déjean, 2003].

Si les expérimentations effectuées ont permis de vérifier l'intérêt de l'architecture proposée, celle-ci reste donc limitée en terme de flexibilité : elle n'est pas minimale (si un agent n'effectue pas certaines opérations, par exemple changement de comportement ou déplacement, tous les micro-composants lui sont quand même associés) et ne permet pas de modeler et de personnaliser la structure des agents. Par exemple, il n'est pas possible de définir un agent immobile ou un type quelconque d'agent doté de mécanismes particuliers. Pourtant, il semble que si le modèle et l'architecture d'agent sont ajustés au besoin, l'adaptation dynamique est simplifiée et l'implémentation est plus efficace (performances d'exécution en temps et en occupation mémoire).

Différents modèles d'agents Afin de complèter ce besoin de flexibilité de l'architecture, nous avons étudié quelques modèles d'agent, en s'intéressant à leurs implémentations et plus précisément à leurs architectures. Dans une démarche de classification, on peut être amené à distinguer les agents suivant différentes capacités (dans la littérature des systèmes multi-agents, on trouve fréquemment le terme de *composantes*) indépendantes et orthogonales, à caractère individuel ou social :

— l'autonomie, déclinée diversement dans différents modèles d'agent et leur implantation, qui induit les notions de cycle de vie et d'activité ;
— le savoir et le savoir-faire (éventuellement offert sous forme de services) ;
— une capacité de décision ;
— l'évolutivité (apprentissage…) ;
— la possibilité d'engendrer de nouveaux agents ;
— la communication ;
— l'interaction avec l'environnement (perception…) dans lequel l'agent est situé (par exemple, un agent mobile est situé géographiquement sur un réseau de machines et interagit avec lui *via* un ensemble des services) ;
— la réactivité [8] ;
— la mobilité ;

8. Un agent est réactif s'il répond de manière opportune aux changements de son environnement issus de stimuli externes. Il n'a pas besoin d'une représentation symbolique élevée de la perception de son environnement.

— ...

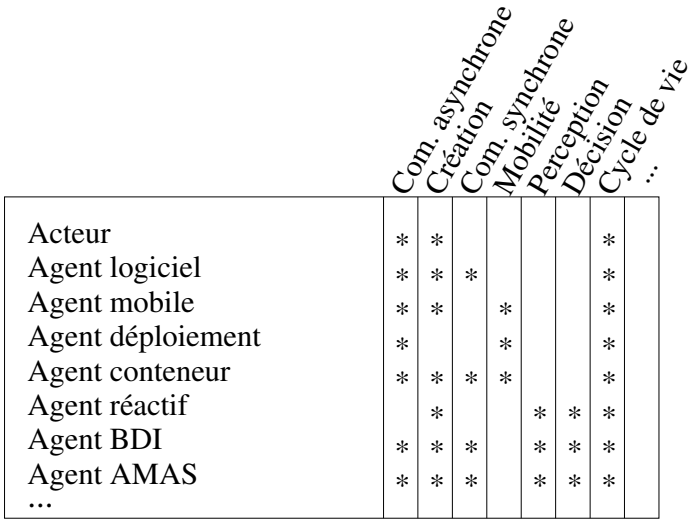

Figure 2.7 – Quelques modèles d'agents et leurs capacités

La notion de cycle de vie se retrouve dans tous les modèles sous des formes différentes (cf. figure 2.7) et la plupart des agents ont besoin de moyens de communication respectant certains standards (invocations de services distribués, KQML, FIPA-ACL...). Les différentes propriétés d'un agent définissent le modèle : par exemple, les agents BDI [Rao and Georgeff, 1995], les agents AMAS, les agents de déploiement [Hall et al., 1997], les agents conteneurs [9], etc.

La mise en œuvre de ces différents modèles est en général liée à une plate-forme spécifique qui permet de simplifier la conception et le développement d'une application agent. On peut citer par exemple AGENTBUILDER [10] pour le modèle BDI ou encore JADE [11] pour concevoir des agents suivant le standard FIPA [12]. De nombreuses autres plates-formes de développement de systèmes multi-agents peuvent être trouvées sur le site suivant : http://www.agentlink.org.

Au sein d'une application, on peut trouver des modèles d'agents différents, y compris des modèles hybrides possédant des composantes issues de différents modèles. On peut s'interroger sur l'intérêt qu'il y aurait (pour faciliter le développement) à pouvoir définir des modèles d'agents ¡¡ à la carte ¿¿. Nous pensons que certains agents pourraient bénéficier de composantes issues d'autres modèles, pour obtenir par exemple des agents BDI mobiles ou des acteurs sans changement de comportement.

L'ambition de notre démarche n'est pas de proposer un modèle et une architecture tellement générique qu'elle permettrait de réaliser ces différents modèles d'agents, cela serait probablement utopique et sûrement moins pertinent que d'employer les plates-formes adéquates pour chaque modèle. Nous expérimentons simplement des architectures suffisamment flexibles pour réutiliser des capacités spécifiques à certains modèles d'agents et permettre la conception de modèles adéquats.

9. Cf. chapitre précédent
10. http://www.agentbuilder.com
11. http://jade.cselt.it
12. http://www.fipa.org

Du modèle d'agent au méta-modèle Nous proposons d'étendre les idées et principes architecturaux énoncés précédemment, en relâchant les contraintes (modèle d'acteur, modèle dynamiquement adaptable, mobilité) et en améliorant les points faibles, en particulier sur la sûreté de l'assemblage des micro-composant.

Au lieu de proposer un unique modèle d'agent, nous essayons d'en abstraire les caractéristiques principales et de travailler au niveau supérieur. Ainsi, en s'inspirant de la vision *ingénierie des modèles* [13], on peut considérer que le modèle d'agent mobile adaptable est au niveau 1 et permet d'engendrer des agents de niveau 0. Pour générer des modèles d'agents différents, nous proposons une sorte de méta-modèle d'agent (niveau 2), dans lequel un ensemble de règles définit ce que nous nommons un *style* d'agent (figure 2.8). L'interface graphique que nous proposons peut être vue comme un langage de modélisation.

Ainsi, l'un de nos objectifs est de vérifier si les principes proposés précédemment que nous reprenons sous le terme de *style d'architecture* (organisation en étoile autour d'un connecteur, séparation réflexive des codes fonctionnels au niveau de base et des codes non-fonctionnels au méta-niveau, utilisation de composants à grain fin pour réifier les services non-fonctionnels au méta-niveau) sont toujours applicables, en allant vers plus de flexibilité. L'idée est de permettre la définition de modèles d'agents par le biais de la construction d'une architecture conforme au style d'architecture des agents mobiles tels que présentés précédemment.

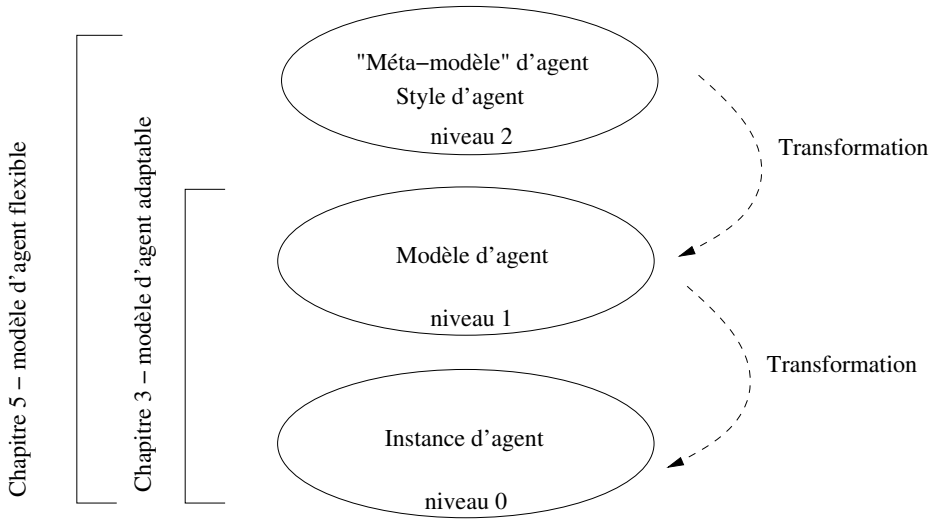

Figure 2.8 – Style d'agent flexible

Besoin de validation Dans l'approche de l'architecture figée précédente, il était possible de valider différentes propriétés soit par typage vérifié lors de la compilation soit lors de l'exécution par des tests. Ces vérifications concernent à la fois le niveau structurel (fonctionnalités nécessaires, micro-composants adéquats) ainsi que le niveau sémantique (cohérence de l'assemblage de micro-composants). Dans la nouvelle approche, les possibilités de modèles d'agent sont infinies. La seule abstraction disponible pour raisonner est le style d'architecture dont les caractéristiques ne sont pas suffisantes (trop abstraites) pour obtenir des propriétés fortes.

13. http://www.omg.org/mof

Il serait pourtant intéressant de pouvoir valider les assemblages de composants (dépendances...), ainsi que certaines propriétés comportementales (terminaison, sûreté, vivacité...). Par exemple, après avoir défini un nouveau modèle d'agent il faudra s'assurer qu'un agent réagira à une perception issue d'un composant d'interaction. Ainsi, il semble nécessaire de proposer des outils de validation qui pourraient être utilisés de manière systématique lors de la définition d'un nouveau modèle d'agent.

Vers un environnement de développement dédié La flexibilité de l'architecture doit permettre non seulement le remplacement dynamique de composants, mais aussi la définition de structures d'agent personnalisées (éventuellement l'ajout de nouveaux composants à une structure existante et symétriquement le retrait). Ainsi, le concepteur doit pouvoir :

- définir un modèle d'agent adapté au besoin applicatif,
- réaliser le modèle d'agent, par assemblage de composants logiciels de grain fin (réutilisés, spécialisés ou nouveaux) qui constituent l'ensemble de ses mécanismes non-fonctionnels,
- valider l'assemblage de micro-composants (cohérence des types, résolution des dépendances...) ainsi que des propriétés comportementales,
- générer un squelette de code de l'architecture choisie pour simplifier et rendre plus fiable l'étape d'implémentation dans laquelle il n'aura plus qu'à intégrer les codes fonctionnels,
- et enfin, programmer la reconfiguration des agents en cours d'exécution en faisant évoluer leurs capacités pour leur permettre de s'adapter dynamiquement à leur environnement d'exécution.

La dernière étape n'est pas du même niveau, l'adaptation recherchée se déroulant dans une phase d'exécution de l'agent. C'est simplement une préoccupation qui ne doit pas être oubliée, car nous cherchons à produire des agents dynamiquement adaptables.

2.1.4.2 Architecture flexible d'agent logiciel

Pour mettre en œuvre les propositions précédentes, nous devons pouvoir :

- décrire des architectures d'agents, donc utiliser un langage de description d'architecture (ADL) ;
- valider l'architecture, donc utiliser un formalisme de vérification basé sur sa description ;
- générer du code et l'optimiser, donc utiliser un mécanisme de génération de code.

Il existe un certain nombre d'outils répondant à ces besoins [Medvidovic, 1996]. Toutefois, le temps d'investissement nécessaire à l'acquisition de ces outils nous a semblé supérieur au développement d'une solution *ad hoc*, sachant que notre objectif est d'obtenir un simple prototype permettant de valider nos propositions. Malgré tout, l'intégration de ces outils reste envisageable pour former une solution plus robuste et plus complète.

La figure 2.9 donne une vision globale du processus présenté dans ce mémoire et constituant une première étape, destinée à valider par prototypage les concepts proposés.

Conception d'une architecture d'agent Le concepteur d'une application agent peut soit réutiliser un modèle et une architecture existante, soit réaliser un nouvel assemblage. La

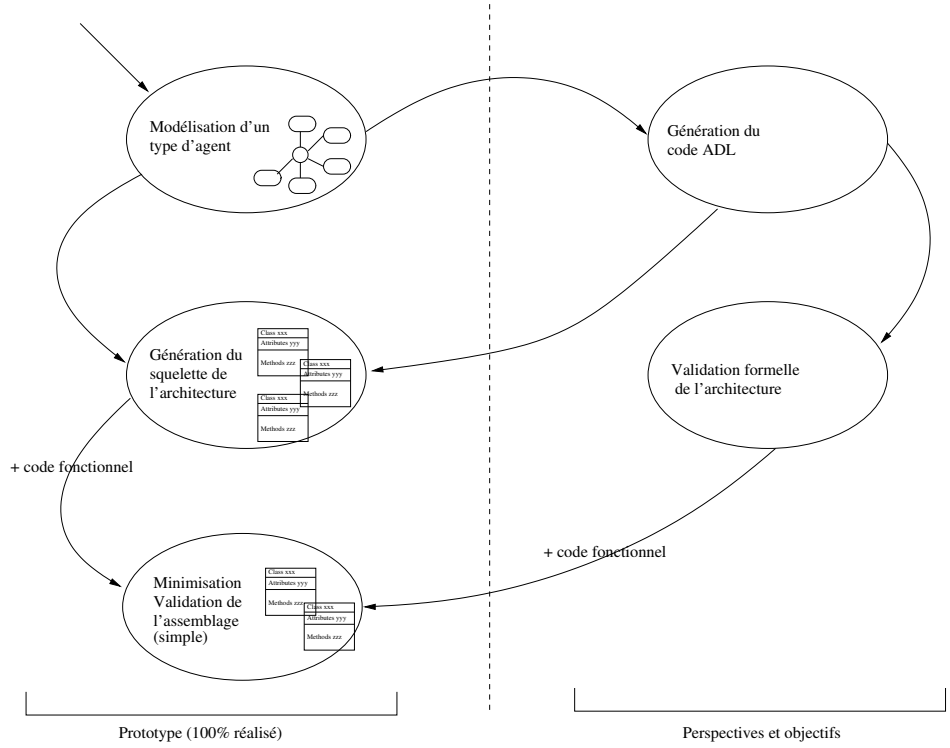

Figure 2.9 – Processus de développement

réutilisation d'une architecture offre un gain de productivité intéressant puisque cela permet de se dispenser d'une phase de validation (réalisée antérieurement). L'architecture choisie peut ensuite être spécialisée pour former un modèle d'agent adéquat. Cette spécialisation se fait par ajout ou retrait (sous réserve d'une nouvelle validation dans ce cas) :

— de micro-composants qui seront intégrés par défaut dans l'architecture de chaque nouvel agent ;
— ou bien de types de micro-composants. Ce qui permet de retarder le choix au moment de la création d'une instance d'agent (pour une configuration adaptée au contexte d'exécution par exemple).

Pour faciliter la réutilisation et le choix des micro-composants, ceux-ci doivent disposer de méta-données décrivant au minimum leur usage en langage naturel, ainsi que de manière plus formelle leur type, les services offerts, les services requis et éventuellement d'autres propriétés utiles. L'assemblage des micro-composants peut se faire de manière graphique si l'on souhaite simplifier et accélérer cette phase de conception.

Principes architecturaux *Différents niveaux de micro-composants*

Dans l'architecture d'agent mobile adaptable présentée précédemment, tous les micro-composants n'ont pas le même statut dans le méta-niveau. Ceux qui offrent leurs services au niveau de base exportent leurs primitives d'accès dans un composant qui fait le lien entre les deux niveaux (classe `QuasiBehavior`, cf. l'exemple de l'envoi de message. Mais certains micro-composants n'offrent des services qu'au niveau méta (par exemple les primitives d'accès au cycle de vie d'un acteur ne doivent pas être accessible depuis le code

fonctionnel), et l'accès depuis le niveau de base ne doit pas être possible. D'autres servent de point d'entrée (micro-composant de réception de message et composant de perception en général) dans l'agent, et dans le cas général, cela n'a pas de sens d'invoquer leurs services depuis l'intérieur de l'agent. Ainsi nous distinguons 3 catégories de micro-composants et des propriétés d'accessibilité pour ces différents cas :

- *Micro-composants de service pour le niveau de base* : accès depuis le niveau de base (envoi de messages, mobilité...),
- *Micro-composant de service pour le méta-niveau* : accès limité au méta-niveau (cycle de vie, analyseur...),
- *Micro-composant d'interaction* : accès limité à l'extérieur de l'agent (réception de messages, perception de l'environnement...).

Ces distinctions permettent de renforcer la sûreté et la sécurité de l'architecture produite. En effet, cela permet de définir une politique de contrôle d'accès garantie par la construction : le code fonctionnel, au niveau de base, n'a pas de visibilité sur le méta-niveau. Par exemple, on ne peut pas perturber l'exécution du thread de l'agent, géré par un micro-composant de service pour le méta-niveau. De même, l'invocation des services d'interaction n'est pas possible pour aucun des micro-composants de l'agent. Ces distinctions n'étaient pas nécessaires dans la version précédente de l'architecture d'agent adaptable, car les mécanismes de protection avaient été intégrés de manière *ad-hoc*.

Style d'architecture

Un agent est composé d'un ensemble de micro-composants distincts, connectés à un connecteur unique.

Il n'est pas possible d'intégrer deux micro-composants de même type, car ils offriraient les mêmes services via des méthodes identiques (par exemple deux composants d'envoi de messages). Au delà du conflit créé à la compilation, il serait anormal d'offrir des services identiques dupliqués sans moyen de les différencier. Ainsi, un concepteur qui aurait besoin de re-définir une architecture d'agent nécessitant plusieurs micro-composants identiques peut toutefois passer par l'utilisation d'un micro-composant composite, agrégeant un ensemble de micro-composants identiques et offrant à l'agent un service spécialisé. Par exemple, un agent possédant deux boîtes aux lettres de messages pourra utiliser un composant composite encapsulant deux micro-composants de boîte aux lettres standards, en offrant des méthodes d'accès différenciées (par ex. `put(Messsage m, int identBAL)` et `Message get(int identBAL)`). On peut toujours définir des méthodes par défaut pour cacher cette duplicité au niveau de base par exemple.

Il n'y a pas de nombre minimal de micro-composants. Toutefois l'absence d'un composant de cycle de vie risque de limiter l'intérêt d'un tel agent. Ce niveau de vérification (plutôt sémantique) pourra être effectué lors des étapes de validations formelles de l'architecture, par la vérification de propriétés adéquates (vivacité, etc.). Concrètement, un agent doit disposer d'un composant de perception et d'un cycle de vie pour permettre un fonctionnement minimal. Il n'y a pas de limite en nombre de micro-composants.

L'interface entre le niveau fonctionnel et les services offerts par les micro-composants est réalisée par un objet spécifique (`QuasiBehavior` dans la figure 2.6), afin de réaliser la séparation des niveaux de services proposée plus haut et de renforcer l'expressivité du code fonctionnel. Tous les services offerts au niveau de base sont mis à disposition du code fonctionnel par cette classe intermédiaire.

Adaptation dynamique de l'architecture Dans le modèle d'agent adaptable, la capacité

d'adaptation dynamique de l'architecture est une caractéristique intrinsèque. Nous proposons de réifier la capacité d'adaptation, en considérant l'adaptation dynamique au même titre que les autres caractéristiques non-fonctionnelles (mobilité...). Ainsi, il est possible de générer des architectures d'agent qui ne soient pas dynamiquement adaptables (donc avec des performances d'exécution légèrement supérieures et une meilleure sûreté de fonctionnement par exemple). Pour rendre un agent adaptable, il faudra intégrer des micro-composants qui gèrent les mécanismes d'adaptation dynamique. En général, un seul composant d'adaptation ne suffira pas : il faudra prévoir un cycle de vie capable de prendre en compte les besoins et les demandes d'adaptation, de manière similaire à la proposition précédente.

Minimisation Lors de l'étape de choix du modèle d'agent, il est possible que l'architecture sélectionnée ne soit pas optimale, c'est-à-dire que certains micro-composants non-fonctionnels aient pu être sélectionnés par le concepteur mais non utilisés en pratique. Cela peut se produire soit dans le cas d'un mauvais choix d'architecture, soit dans le cas où le concepteur est parti d'un type générique prédéfini d'agent qui s'avère un peu trop complet[14]. Par exemple, si le concepteur a réutilisé l'architecture d'acteur mobile pour implémenter un algorithme de calcul parallèle qui n'exploite pas la mobilité, la minimisation permet de suggérer le retrait du micro-composant de mobilité.

Pour cela, nous proposons de dériver du code fonctionnel de l'agent un type appelé *type réduit* (ou type d'implantation), qui correspond à un ensemble de types de composants effectivement utiles (ou encore aux mécanismes strictement nécessaires à l'exécution). A partir du type réduit ainsi identifié, il sera possible d'obtenir une architecture d'agent *ad hoc*, minimale en terme de nombre de composants.

Cette minimalité de l'architecture a des avantages en terme d'efficacité à l'exécution (d'autant plus si les agents se déplacent sur le réseau), mais aussi en termes de sûreté et de sécurité car la vérification et la validation des assemblages (dépendances...) sont facilitées. Par ailleurs, il n'est pas nécessaire d'expliciter les services de méta-niveau (mécanismes d'exécution) requis par le code fonctionnel puisqu'ils peuvent être directement déduits du code.

Dans le cas le plus particulier d'un agent qui pourrait changer de comportement fonctionnel (cas des acteurs), le type réduit pourrait être représenté par un diagramme états-transitions (cf. figure 2.10) : les transitions correspondraient aux changements de comportement et chaque état représenterait le type réduit pour un comportement.

14. Précisons que dans le cas inverse (pour un micro-composant manquant), l'erreur sera détectée lors de l'intégration du code fonctionnel, via une erreur explicite de compilation dans notre prototype.

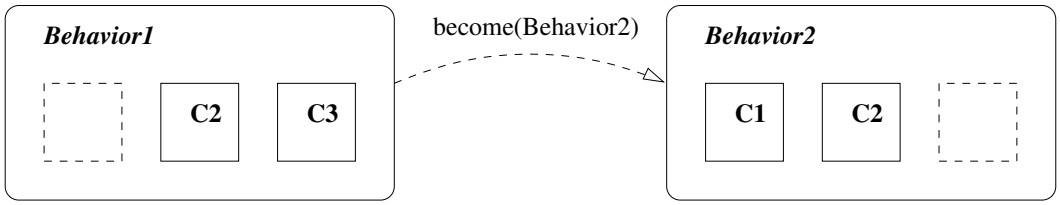

Figure 2.10 – Type réduit d'agent, représenté par un diagramme état/transition

Problèmes ouverts Le problème de la "mutation" des agents, *i.e.* le changement dynamique de modèle, reste ouvert. Au delà d'un besoin qui reste à être démontré, nous pensons qu'il serait préférable de ne pas autoriser la mutation si on veut obtenir des systèmes fiables et vérifiables a priori (on pourrait cependant autoriser les mutations prévues lors de la conception).

L'adoption dynamique par un agent d'un micro-composant d'un type non prévu ne nous semble pas justifiée (un agent pourrait découvrir par lui-même la possibilité de "se bonifier" en changeant ses mécanismes opératoires mais, à notre sens, cela ne pourrait résulter que de capacités d'introspection beaucoup plus élevées que celles que nous envisageons aujourd'hui). En effet, pour utiliser à bon escient un composant découvert dynamiquement, il faudrait que l'agent puisse comprendre la sémantique d'utilisation de ce composant, puis de raisonner sur l'apport de ce composant dans son architecture avant de pouvoir l'y ajouter.

Néanmoins, il est envisageable qu'un agent puisse acquérir dynamiquement un comportement fonctionnel nouveau (dans le cas d'une adaptation fonctionnelle) qu'il aura, par exemple, reçu par message (évolution par changement de comportement dans le cas des acteurs de K. Hewitt et G. Agha). Pour que ce comportement soit effectivement exécutable, il faut que les micro-composants opératoires nécessaires soient disponibles au méta-niveau ; s'ils ne le sont pas, il faut reconfigurer le méta-niveau à partir de nouveaux micro-composants et redéfinir ainsi dynamiquement le modèle d'agent. On peut alors envisager l'intégration dynamique de micro-composants standards (création d'agents, communication, mobilité…).

2.1.5 Macro-architecture : patron de conception pour les systèmes P2P

La conception des systèmes à grande échelle et à topologie instable est complexe et que de nombreuses préoccupations comme la gestion de l'hétérogénéité ou des volumes de données, la volatilité, les performances doivent être considérées. Du point de vue génie logiciel, pour simplifier la conception et la mise en œuvre de tels logiciels et permettre leur réutilisation, il est nécessaire d'employer des technologies adéquates. Nous avons vu que les systèmes P2P purs décentralisés sont les mieux adaptés à la mise à disposition de ressources volatiles ou évolutives et aux systèmes à grande échelle et à topologie instable. Pour cette raison, ce modèle d'organisation est l'élément de base des propositions faites dans ce chapitre.

2.1.5.1 Patron de conception

Un patron de conception (*Design Pattern* [Gamma et al., 1995] ou canevas logiciel) décrit une solution standard pour répondre à un problème d'architecture et de conception logicielle, afin de réduire le temps de conception et d'augmenter la qualité du résultat.

Eléments constitutifs d'un système P2P Dans cette section, nous proposons d'identifier les différentes fonctionnalités des systèmes P2P décentralisés et d'en déduire l'ensemble des composants d'une architecture générique de ces systèmes. Nous entendons par *ressource* non seulement les éléments physiques (fichier, base de données, périphérique) mais aussi l'ensemble des services offerts sur les différents sites.

Outre les fonctionnalités d'adhésion volontaire au réseau et de retrait, les pairs doivent au minimum pouvoir :

— mettre des ressources à disposition de la communauté (publication),
— rechercher une ressource à partir d'une description,
— exploiter une ressource trouvée.

1. La publication d'une ressource consiste à l'insérer dans le réseau et à permettre à d'autres pairs d'y accéder. Au plus simple, la ressource est hébergée localement chez le propriétaire, mais elle peut aussi être hébergée ailleurs sur le réseau. Un composant de **publication** est donc nécessaire, et du côté de l'hébergeur un mécanisme doit permettre la publication de ressources par un tiers.
2. Au contraire des systèmes structurés qui proposent un ou plusieurs serveurs d'index, la localisation dans les systèmes P2P purs ne passe pas par l'utilisation de serveurs. Pour accéder aux ressources, il y a donc au départ un besoin de découverte et de localisation des pairs serveurs, puis ensuite, sur ces pairs, un besoin de fouille locale pour l'extraction des ressources et services. Pour cela nous distinguons deux composants au sein du mécanisme de recherche : l'un de localisation ou de **recherche globale** pour la découverte et l'accès aux pairs, l'autre pour la **recherche locale** sur le pair serveur.
3. En complément à la localisation, le système doit permettre à un pair de découvrir des méta-informations sur le système (sur les pairs et les ressources) et de maintenir un ensemble de connaissances sur le réseau P2P. Ces connaissances sont exploitées pour optimiser les recherches futures (par exemple par inondation). Pour cela, un élément de **méta-information** répertorie les connaissances des pairs sur eux-mêmes et sur la topologie du réseau P2P, qui peut évoluer en permanence (associations type de ressources/localisation, pairs voisins/ressources, etc.).
4. Après avoir découvert une ressource, il faut pouvoir l'exploiter. Il peut s'agir par exemple de communiquer à l'utilisateur la description (méta-données) de la ressource trouvée, d'exécuter un service puis de transmettre les résultats, de lancer une nouvelle recherche si la ressource dépend d'une autre, ou plus simplement de télécharger un fichier. Ces opérations sont à la charge d'un composant dédié à **l'exploitation des ressources**. Celui-ci doit également superviser l'exploitation de la ressource (pour adapter par exemple l'exploitation aux conditions d'exécution : variations de bande passante, d'occupation du processeur...).
5. De plus, dans un contexte de ressources volatiles et évolutives, un composant supplémentaire fournissant un moyen de s'interfacer de manière uniforme avec les

ressources (accès à une base de données, à un système de fichiers, à un *Web service*...) peut être ajouté. De cette façon, si la forme d'une ressource change (nouvelle version) seul le composant d'**accès aux ressources** doit être modifié. Des systèmes comme Spitfire [15] ou JDBC [16] peuvent être employés.

Il résulte de cette décomposition un ensemble de briques de bases pour la construction de systèmes P2P, que l'on peut implémenter sous forme de composants logiciels : publication, localisation, recherche locale, information, accès aux ressources et exploitation. Il se pose alors le problème de leur déploiement. En effet, pour augmenter les capacités de personnalisation des applications, certains de ces composants logiciels sont fournis par le client. Ils doivent donc être installés sur les sites serveurs. Ainsi, nous proposons dans les sections suivantes un modèle d'architecture distribuée à base d'agents et de composants.

Problématique du déploiement Dans une première étape, les composants doivent être transportés sur les sites serveurs, pour y être déployés. Leur déploiement est constitué de différentes activités de mise et de maintien en production : installation, configuration, activation mais aussi la maintenance (adaptative et préventive) et mise à jour (voir [Carzaniga et al., 1998c] pour une définition générale des activités de déploiement). Pour être activés et fonctionnels, ils doivent s'exécuter dans un environnement adéquat qui leur fournit un ensemble de services (communication...) ainsi que des ressources d'exécution (temps processeur, mémoire...).

Le déploiement de composants classiques nécessite généralement l'usage de méta-données spécifiques, appelées *descripteurs de déploiement*, souvent fournies sous forme de fichiers XML ou de scripts spécifiques.

Dans les sections suivantes, nous proposons de réaliser ces différentes étapes, en encaspulant les composants dans des agents mobiles. En effet, ceux-ci, via leur mobilité et leur pro-activité, peuvent localiser des ressources dans des contextes de réseaux instables, et distribuer des composants sur les sites serveurs. De plus, si les composants s'exécutent à l'intérieur de l'agent, ils pourraient bénéficier des services non-fonctionnels des agents (communication asynchrone, mobilité pro-active...). Enfin, le déploiement d'un composant par une entité active et autonome permet de simplifier voire de supprimer les méta-données de déploiement, ce qui simplifie cette étape.

15. `http://edg-wp2.web.cern.ch/edg-wp2/spitfire`, utilisé dans le projet DATAGRID pour l'accès uniforme aux bases de données
16. Connecteur d'accès à différentes bases de données depuis le langage Java.

Un modèle pour le déploiement de composants Notre architecture répartie se distingue des solutions classiques par la décentralisation et l'utilisation d'agents mobiles adaptables (présentés au chapitre précédent) pour supporter le déploiement des composants logiciels.

Déploiement des composants par les agents (macro-architecture)

Dans cette proposition, les composants métiers prennent la forme de comportements d'agents. Cela revient à encapsuler les composants métiers dans des agents mobiles, à raison d'un agent par composant. De cette manière le concepteur bénéficie des avantages du modèle de programmation agent (abstraction, expressivité…). De plus, le composant peut accéder à tous les mécanismes fournis par les agents : autonomie, communication asynchrone, mobilité, adaptation dynamique… En particulier, la mobilité permet de déplacer à volonté un composant métier, afin de le rapprocher des traitements ou d'optimiser son fonctionnement.

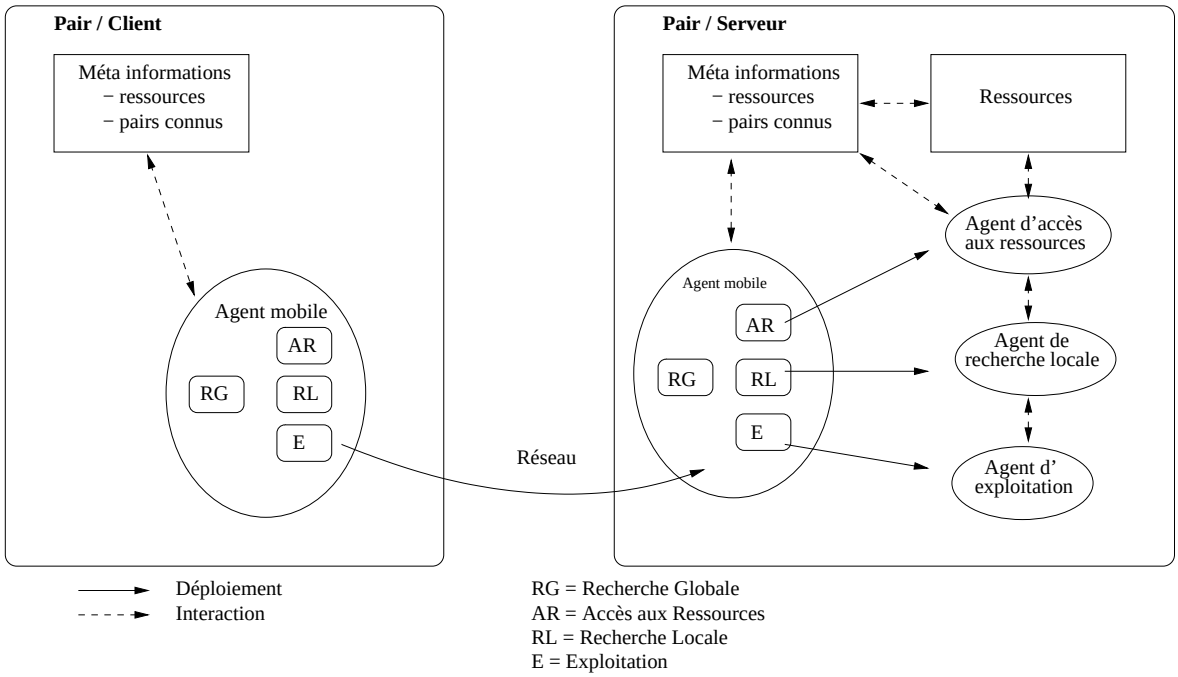

Figure 2.11 – Déploiement des composants

Le principe du déploiement est schématisé dans la figure 2.11 (pour une meilleure lisibilité, nous avons distingué le pair jouant le rôle de client de celui jouant le rôle de serveur mais ils sont bien sûr complètement symétriques) :

— Les agents mobiles assurent d'abord le **déploiement géographique adaptatif** des composants. Pour chaque requête d'un client, un agent mobile (ou plusieurs pour exploiter le parallélisme) transporte les composants sur le réseau. Le composant de localisation (ou recherche globale, RG) constitue l'essentiel du comportement de l'agent mobile. Celui-ci se déplace de pair en pair en effectuant dynamiquement la localisation, ce qui lui permet d'adapter la recherche à l'état courant du système P2P.

— Lorsque l'agent mobile arrive à destination, il installe un mini système multi-agent à partir des composants (AR, RL et E), pour traiter la requête localement. Là, chaque composant est encapsulé dans un agent dont il constitue le comportement et à travers lequel il offre ses services. L'agent sert de **conteneur** et d'**activateur** du composant

et lui fournit les mécanismes non fonctionnels (communication...) nécessaires à son exécution.

— Les agents ainsi déployés coopèrent pour réaliser la tâche de recherche ou d'exploitation des ressources suivant une algorithmique client, ce qui permet une forte personnalisation de ces activités.

Adaptation pour les composants au sein des agents (micro-architecture)

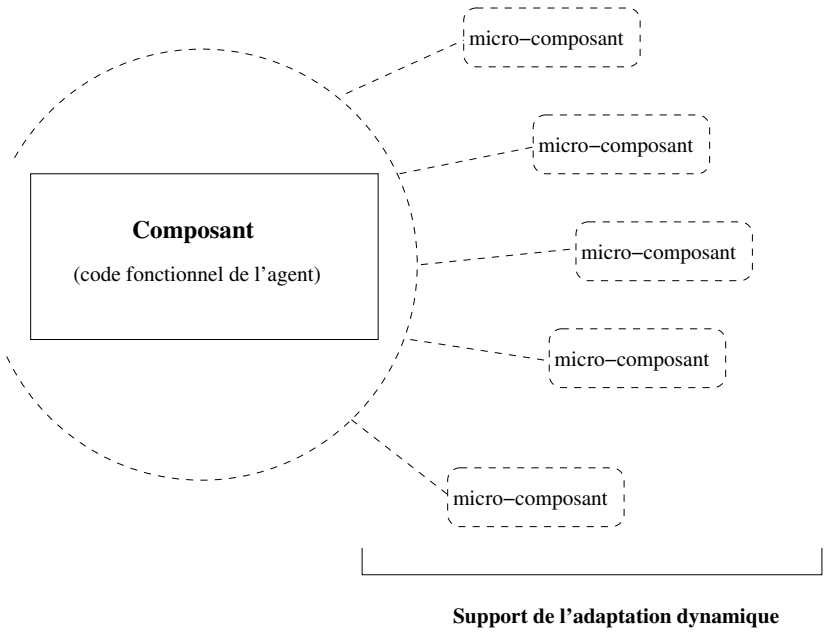

Figure 2.12 – Environnement d'exécution adaptable

Pour augmenter l'adaptation et la robustesse des applications à base d'agents face aux problèmes d'échelle et en particulier à ceux liés à la sûreté de fonctionnement, nous avons proposé dans le chapitre précédant un modèle d'agent auto-adaptable. Nous nous appuyons sur cette micro-architecture d'agent pour adapter l'environnement d'exécution du composant. La réutilisation du modèle d'agent pour déployer des composants permet de faire bénéficier à ces composants des propriétés d'adaptation des agents (figure 2.12). On peut, par exemple, adapter le mécanisme de communication du composant d'exploitation (chiffrement, compression...) en redéfinissant le micro-composant de communication de l'agent d'exploitation.

Ainsi, l'adaptation est réalisée au plus près de l'application, en restant transparente pour le programmeur du composant métier. L'agent joue le rôle d'un intergiciel adaptable pour chaque composant et l'association d'un agent à un composant permet une finesse d'adaptation plus importante que dans un environnement d'exécution de composant traditionnel. On peut adapter un seul agent sans que les modifications aient d'impact sur les autres agents s'exécutant dans le même système d'accueil. Cela contribue à la flexibilité globale de l'application.

Apports mutuels Agents-Composants-P2P Il est possible de faire une analogie entre le modèle composant-conteneur et notre modèle d'agent adaptable, particulièrement si l'on considère la séparation des préoccupations. D'une part, les composants comme les comportements des agents représentent le code fonctionnel (code métier). D'autre part, le conteneur

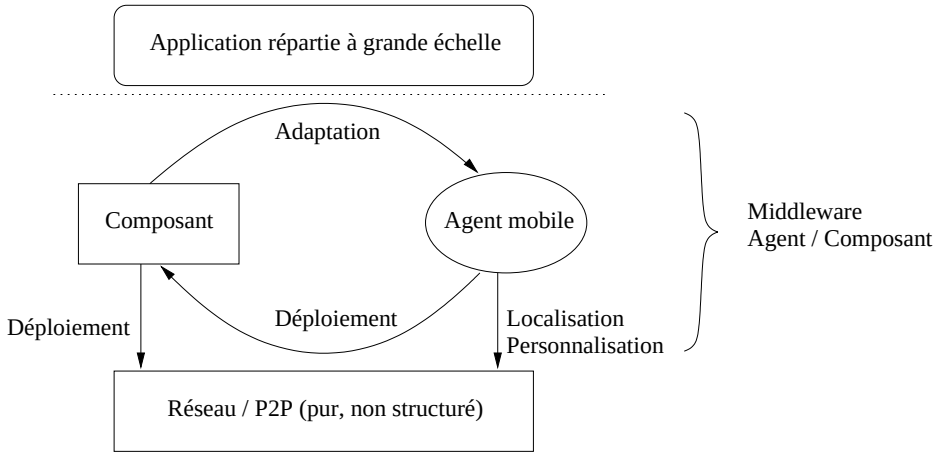

Figure 2.13 – Complémentarité des technologies

permet au composant d'instancier, d'activer et d'accéder aux services non fonctionnels fournis par l'environnement d'exécution (par exemple la gestion du cycle de vie, de la sécurité ou des communications). Dans notre modèle, l'agent joue le rôle de conteneur : ses micro-composants enveloppent le comportement fonctionnel et implémentent les mécanismes non-fonctionnels (envoi des messages, réception, cycle de vie…) ou éventuellement les délèguent au système d'accueil. Dans son rôle de conteneur, l'agent supporte donc l'adaptation du composant (configuration et reconfiguration).

Pour déployer les composants, nous bénéficions donc des avantages des agents en termes de mobilité et d'adaptation : on peut déplacer et adapter le composant via l'agent qui le contient. Notre architecture d'agent adaptable permet d'obtenir la flexibilité et l'extensibilité nécessaire à l'adaptation : adapter un agent revient à changer un ou plusieurs de ses micro-composants. L'agent est ainsi le **vecteur de l'adaptation individuelle et du déploiement** des composants. Cette forme d'adaptation est complètement séparée de la programmation des composants, ce qui contribue à simplifier le développement et la réutilisation.

La mobilité et l'adaptabilité des agents jouent donc un rôle essentiel dans la phase de déploiement. Agents et composants [17] constituent une sorte de *middleware* permettant aux applications l'accès transparent au réseau P2P. Nous résumons dans la figure 2.13 la complémentarité des technologies évoquées précédemment (une flèche de A vers B indique un apport de la technologie A à la technologie B).

Utilisation du patron de conception Pour utiliser le patron de conception, il est nécessaire de réutiliser ou adapter les composants qui seront embarqués dans les agents. La figure 2.14 présente les différents points d'adaptation fonctionnelle ; quelques exemples sur la manière de les adapter sont décrits ci-dessous.

Adaptation de la recherche globale - A_4

Par nature, la mobilité d'agent est *proactive* : c'est l'agent mobile lui-même qui décide de ses déplacements de manière autonome. Dans notre cas, l'agent de recherche s'appuie sur une politique personnalisée qui peut faire intervenir une évaluation de la qualité des

17. On trouvera une étude sur les apports mutuels entre agents et composants dans [Briot, 2004].

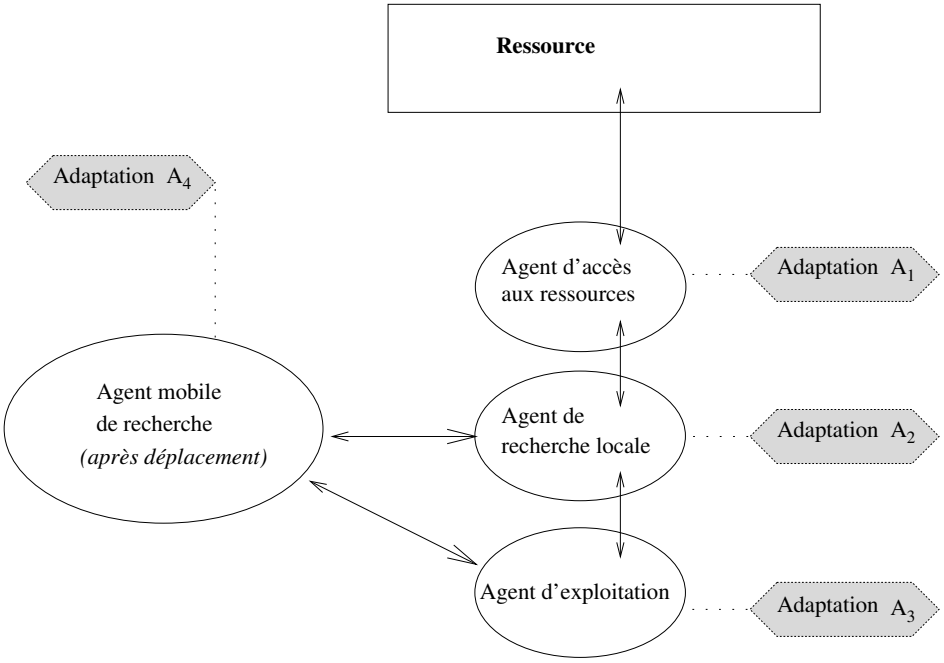

Figure 2.14 – Les différents points de spécialisation

résultats trouvés, l'état du réseau, voire les connaissances du site serveur sur le reste du domaine. Il peut arrêter la recherche (si les résultats trouvés sont satisfaisants), la poursuivre sur un autre site, retourner sur le site client, éventuellement relancer une nouvelle recherche locale, etc. Ainsi, le chemin de l'agent mobile peut être construit dynamiquement et c'est au plus près des données que la décision de déplacement est prise.

Dans nos prototypes (et en particulier dans l'implémentation du composant de recherche globale), un mécanisme complémentaire d'acquisition dynamique d'informations sur le domaine a été ajouté. En parcourant le réseau et en explorant des zones initialement inconnues du client, il est possible pour ce dernier de glaner des connaissances sur les sites visités lors de la phase de recherche. Ces connaissances permettent de mettre à jour les bases de données qui, du côté du client, répertorient les propriétés des serveurs distants connus et alimentent les (futures) sélections de serveur. Plus le client dispose d'informations sur d'autres sites, plus les chances de trouver l'information recherchée augmentent. Le protocole de recherche présenté ci-dessous permet l'acquisition dynamique d'informations par le client, *via* un simple envoi de message de l'agent au client.

Adaptation de la recherche locale - A_2

Pour effectuer une recherche non standard spécifique au client, il est habituel de déplacer les données sur le site du client. En cas de gros volumes de données, la solution alternative consiste à déplacer le code de la recherche sur le site serveur. Dans notre proposition, l'agent mobile de recherche se déplace avec un composant de recherche spécialisé. Ce composant, qui implémente l'expertise du client, est un comportement à partir duquel un agent est créé sur le site serveur. Son autonomie permet de réagir dynamiquement aux résultats obtenus et d'orienter le processus de recherche en conséquence (en soumettant une requête affinée par exemple).

Ainsi, on personnalise le processus de recherche tout en limitant les volumes déplacés

sur le réseau (hors l'agent lui-même) aux résultats (et non plus à toutes les données nécessaires au calcul du résultat).

Outre la mise à jour des bases de données du client sur les serveurs répartis (qui alimentent la sélection de serveur), on peut noter l'apport indirect de la mobilité d'agent à la pertinence des résultats : le volume de données accessibles n'est plus limité par les contraintes liées au réseau et par conséquence, la recherche peut donc s'effectuer sur des données plus complètes.

Spécialisation de l'exploitation des résultats - A_3

De la même manière que précédemment, à partir d'un composant spécialisé pour l'exploitation des résultats et *embarqué* au sein de l'agent mobile de recherche, on peut créer un agent d'exploitation sur le site serveur. En fonction des résultats obtenus et du réseau (de ses propriétés et de son état), l'agent d'exploitation peut décider de transmettre les informations obtenues (ou une partie d'entre elles) à l'état brut ou compressées, de transmettre seulement leurs descripteurs, de crypter les informations transmises, etc. Les résultats peuvent également ne pas être transmis, mais conservés par l'agent mobile de recherche lui-même.

Ici, les avantages résident dans l'autonomie des agents et leur capacité à traiter l'information sur le site où elle se trouve.

Adaptation de l'agent à l'hétérogénéité des serveurs - A_1

On fait l'hypothèse que si un serveur est connu d'un client, alors le client sait également comment interagir avec lui. Pour cela, sur le site client, l'application dispose d'un composant d'accès aux ressources avec chaque serveur connu. Ce composant est *embarqué* au sein du comportement de l'agent mobile afin de servir d'interface avec le serveur sur le site distant. Lorsqu'une nouvelle place est découverte dynamiquement, le site client doit acquérir non seulement l'adresse de la place mais aussi le composant d'accès aux ressources avec le serveur. Ceci implique que les places doivent offrir une fonctionnalité supplémentaire pour fournir ce composant.

En pratique, le composant d'accès aux ressources est un comportement d'agent : il contient tous les éléments utiles (script d'interaction et données non fonctionnelles) connus statiquement sur le site client. Sur le site distant, l'agent mobile de recherche crée dynamiquement un agent de dialogue en lui donnant ce comportement (installation) et lui transmet par message la requête et ses paramètres.

Avec un tel protocole, lorsqu'un serveur évolue, les informations connues sur celui-ci par d'autres sites deviennent obsolètes. Ainsi, il se peut qu'un agent de recherche se déplace sur une place avec un composant de communication périmé. Dans ce cas, l'agent peut acquérir *in situ* et à la volée le nouveau composant d'accès aux ressources [18] (et le donner au client pour mémorisation).

De manière générale, les composants peuvent être fournis par le pair client, par le pair serveur ou par un tiers. Ainsi, le composant d'accès aux ressources peut ne pas être systématiquement fourni par le serveur. Mais en cas de mise à jour côté serveur (évolution du mécanisme d'accès aux ressources), il faut prévoir un protocole permettant l'acquisition *in situ* et à la volée du nouveau composant d'accès. Ceci contribue à la robustesse de la recherche et permet l'adaptation dynamique aux évolutions des serveurs.

18. Cette stratégie élémentaire peut ne pas satisfaire les besoins de sûreté ou de sécurité. Au besoin, elle peut être affinée. Dans tous les cas, la stratégie choisie résulte d'un compromis entre les contraintes de performance, d'efficacité, de sûreté et de sécurité.

L'utilisation d'un agent de recherche mobile permet donc l'adaptation dynamique de l'agent aux évolutions du serveur, sans que les opérations de maintenance du serveur ne dépassent le cadre local.

2.1.6 Synthèse

La prise en compte des besoins de flexibilité dans les applications réparties ouvertes à grande échelle augmente la complexité du développement, du déploiement et de la maintenance. Afin de contribuer à la maîtrise de cette complexité, nous proposons des modèles d'architectures logicielles basés sur les technologies agent logiciel, composant logiciel et modèle pair à pair dont nous exploitons la complémentarité. Ces architectures permettent l'adaptation dynamique des aspects non fonctionnels des applications à leur contexte d'exécution ainsi que la personnalisation (l'adaptation fonctionnelle est supportée au niveau des agents, en partie grâce à la mobilité logicielle -mobilité d'agent et de code-).

Les solutions s'appuient sur une décomposition des agents en composants de grain fin et respectent les principes d'autonomie et de séparation des préoccupations et des niveaux. Par ailleurs, elles semblent compatibles avec la prise en compte des problèmes de sécurité. Les expériences menées se sont montrées encourageantes sur le plan de la simplicité d'utilisation et de la réutilisation.

2.1.6.1 Au niveau micro (agent adaptable)

Nous proposons un modèle d'agent logiciel mobile basé sur le modèle de programmation par acteur. Son architecture dynamiquement reconfigurable est constituée d'un ensemble de micro-composants qui implémentent des services non-fonctionnels. L'organisation des micro-composants en étoile autour d'un connecteur central permet d'adapter dynamiquement ces services en fonction du contexte d'exécution [Leriche and Arcangeli, 2004].

Nous avons conçu une implémentation complète de ce modèle, sous la forme d'un prototype appelé JAVACT$^{\delta}$ [Leriche and Arcangeli, 2004], ainsi qu'une implémentation simplifiée (sans analyseur et sans sonde dans le système d'accueil), JAVACT 0.5 [Arcangeli et al., 2004]. Cette version et le *plugin Eclipse* qui permettent d'accélérer et de simplifier le développement sont diffusés sous la forme de logiciels libres [19] (licence LGPL).

2.1.6.2 Au niveau micro (agent flexible)

L'utilisation de micro-composants (granularité fine à l'échelle d'un service unique) au méta-niveau, l'organisation en étoile autour d'un connecteur et l'interface entre les niveaux constituent un style d'architecture compatible avec la proposition du modèle d'agent adaptable.

Conformément à ce style d'architecture, nous proposons un modèle d'agent flexible qui permet d'engendrer différents modèles d'agents dynamiquement adaptables, par assemblage de micro-composants non-fonctionnels réutilisables. Ainsi les agents peuvent bénéficier, *via* un ensemble de micro-composants adéquats, de propriétés de mobilité ou d'adaptation dynamique au contexte d'exécution [Leriche and Arcangeli, 2006]. Ce modèle

19. http://www.irit.fr/recherches/ISPR/IAM/JavAct_fr.html

est conçu pour supporter des analyses et des vérifications à partir d'une transformation de la description d'une architecture d'agent dans un langage de description d'architecture (ADL) et d'outils dédiés aux vérifications.

Nous proposons, sur la base de ce modèle d'agent flexible, une implémentation contenant un environnement de modélisation graphique d'architecture d'agent, une bibliothèque de micro-composants non-fonctionnels et d'architectures prédéfinies, un générateur de squelette d'architecture et un outil de minimisation des architectures prenant en compte les besoins fonctionnels.

2.1.6.3 Au niveau macro

L'analyse d'un certain nombre d'exemples applicatifs a mis en évidence une architecture logicielle générique à base d'agents mobiles adaptables et de composants logiciels, que nous avons abstraite sous forme d'un patron de conception [Leriche and Arcangeli, 2005].

L'utilisation d'agents mobiles simplifie la phase de localisation des ressources dans les systèmes P2P, et en particulier dans les systèmes P2P purs. Nous proposons également d'utiliser les agents mobiles pour personnaliser la phase d'exploitation des ressources, par déploiement d'un comportement adéquat fourni par le client [Leriche et al., 2004].

Plus généralement, nous proposons d'employer des agents mobiles adaptables comme support de déploiement pour les comportements (composants fonctionnels) : d'une part la mobilité d'agent permet un transport adaptatif des composants fonctionnels sur le réseau, d'autre part le méta-niveau joue le rôle de conteneur pour le composant fonctionnel, en lui fournissant les services nécessaires à son exécution et permettant sa configuration (choix des micro-composants) ainsi que sa reconfiguration dynamique [Arcangeli et al., 2006]. Aussi, on peut considérer que le modèle d'agent mobile adaptable avec ses capacités de déploiement constitue un modèle de composant pour les applications réparties à grande échelle.

En pratique, le patron de conception a été mis en œuvre sous la forme d'un *framework* codé en Java à partir duquel nous avons dérivé différents prototypes dont une application pour la mutualisation de ressources en P2P.

2.2 Déploiement des systèmes répartis complexes

2.2.1 Problématique et motivations

Le déploiement de logiciel est un processus complexe qui comprend toutes les activités entre la production du logiciel et sa désinstallation des sites de déploiement [Dearle, 2007b]. Un cycle de vie de déploiement générique comprend l'installation, la désinstallation, l'activation, la désactivation, la mise à jour et la reconfiguration du logiciel [Carzaniga et al., 1998b].

Dans cette section nous nous intéressons au déploiement de logiciels dans des infrastructures réparties à grande échelle qui peuvent être dynamiques telle que les systèmes ubiquitaires, les systèmes P2P. Ces systèmes sont caractérisés par la mobilité des sites et des changements fréquents de la topologie du réseau. Ils sont aussi caractérisés par un grand nombre de machines hétérogènes dotées d'environnements matériels et logiciels différents. Plusieurs outils de déploiement à grande échelle existent, tels que software Dock [Hall et al., 1999], DeployWare [Flissi et al., 2008b], D&C [Group, 2006] et plus récemment KALIMUCHO [Louberry et al., 2011b]. Ces outils sont généralement utilisables dans des topologies réseau fixe et ne prennent pas en compte les situations de déconnexions et les défaillances de machines (ou des liens du réseau) qui caractérisent les environnements ouverts.

Dans [Matougui and Leriche, 2011], nous avons proposé nos premières idées sur l'architecture de notre middleware de déploiement et deux scénarios de déploiement où les outils de déploiement logiciel actuels ne sont pas pertinents. Le premier scénario décrit le déploiement d'une application dans un environnement ouvert avec une topologie imprévisible et des machines qui ne sont pas connues au début du processus de déploiement. L'exemple consiste à exploiter le maximum d'ordinateurs et de téléphones portables connectés en wifi dans une salle de conférences et d'y déployer dynamiquement une application qui montre en temps réel des statistiques sur les machines disponibles. Le deuxième scénario présente le déploiement d'un logiciel de simulation sur une grille de calcul.

Ces deux scénarios nous ont permis de mettre en évidence les spécificités et les problèmes rencontrés lors de déploiement en environnement instable et à grande échelle. La première spécificité, consiste à la nécessité de l'utilisation d'un service de découverte de réseau. En effet, dans les systèmes ouverts, l'administrateur de déploiement ne connaît pas forcément à l'avance les hôtes cible de déploiement que nous devons détecter afin de pouvoir déployer notre logiciel. La seconde spécificité consiste à la satisfaction du problème d'administration multiple, après la détection des sites il nous faut obtenir les droits d'accès aux machines pour permettre le déploiement de logiciel. La dernière spécificité, consiste à prévoir des outils de reconfiguration dynamique et d'autoadaptation pour pouvoir traiter les situations de pannes de machines et les déconnexions.

A partir de ces deux scénarios, nous avons pu conclure qu'une plate-forme de déploiement qui peut répondre aux spécificités et problèmes de ces environnements doit être capable de :

1. détecter, gérer et accéder aux sites cibles d'une manière automatique.
2. gérer les hétérogénéités logicielles et matérielles des sites détectés.
3. fournir un moyen pour la déclaration des dépendances logicielles, les préférences

matérielles et les contraintes de déploiement.

4. calculer un plan de déploiement qui satisfasse les contraintes de déploiement d'une manière automatique.
5. d'exécuter les activités de déploiement avec un minimum d'intervention humaine.
6. d'offrir des mécanismes d'adaptation automatique au moment de l'exécution pour prendre en charge les situations de panne de machines et des déconnexions.
7. s'exécuter dans des topologies dynamiques à grande échelle.

2.2.2 j-ASD : vers un environnement de déploiement autonomique

Dans cette section, nous présentons l'architecture de notre middleware pour le déploiement autonomique de logiciel qui répond aux exigences présentées dans l'introduction. L'architecture proposée est illustrée dans la Figure 1. Le middleware est composé des éléments logiciels suivants :

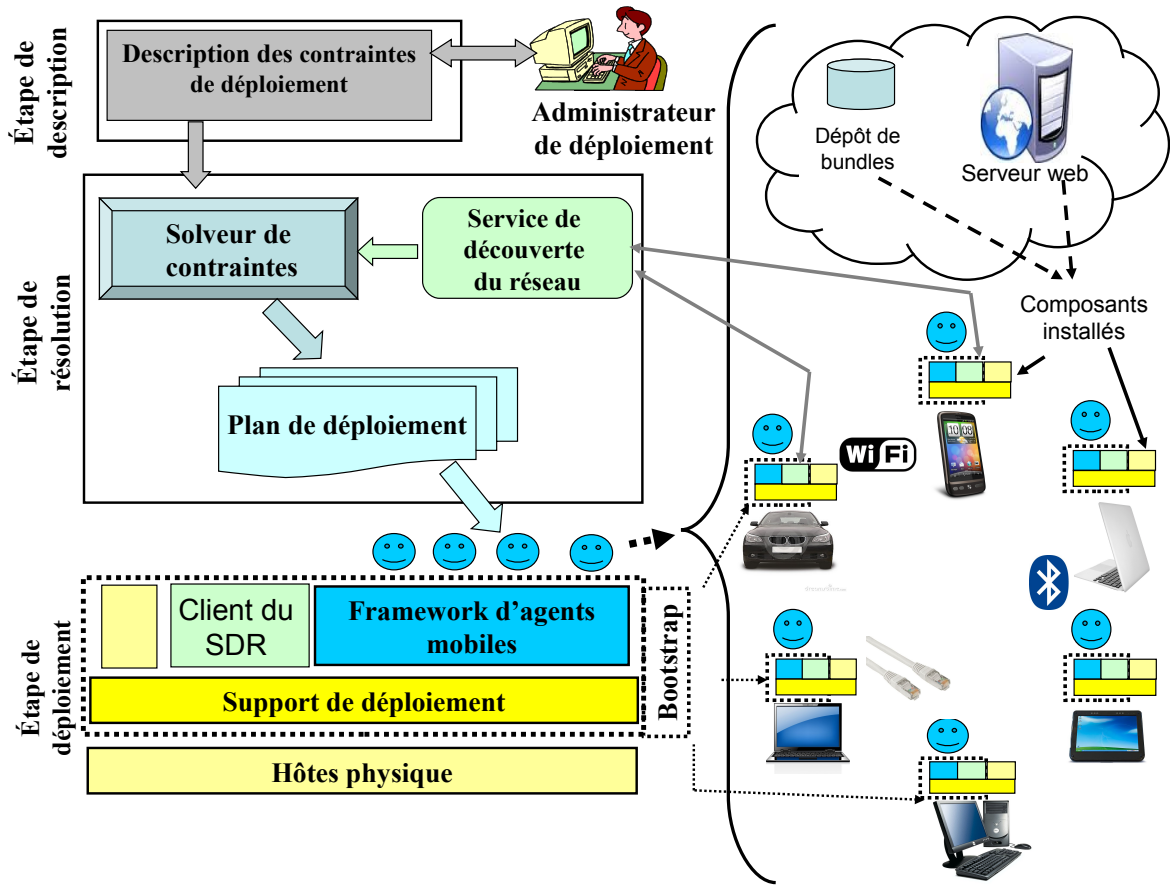

Figure 2.15 – Architecture de j-ASD

Un langage dédié (DSL) à la description des contraintes de déploiement qui permet d'exprimer les contraintes de déploiement et quelques informations sur le logiciel à déployer. Le DSL comporte aussi un parseur et un solveur de contraintes qui permet de calculer un plan de déploiement initial.

Un service de découverte du réseau pour permettre la détection automatique des hôtes cibles de déploiement dans le réseau. ouverts tels que les systèmes ubiquitaires sera possible. Un bootstrap qui permet la préparation de l'environnement d'exécution dans les sites cibles de déploiement. Il a pour but la résolution des problèmes d'administration multiples et des droits d'accès aux sites cibles de déploiement. dépendances logicielles de notre système de déploiement. Un support de déploiement équipé d'un environnement d'exécution, capable de s'exécuter dans des infrastructures matérielles hétérogènes et qui permet d'exécuter tout ou une partie des activités de déploiement.

L'intergiciel est doté d'une interface graphique qui permet aux administrateurs de déploiement à tout moment de vérifier l'état du processus de déploiement et d'intervenir dans une ou plusieurs activités de déploiement.

Enfin, un système d'agents mobiles adaptable prend en charge l'exécution et la supervision du processus de déploiement.

2.2.2.1 Langage de description des contraintes de déploiement

Configurer et déployer des logiciels à grande échelle est une tâche très complexe. La complexité est due à la multitude de composants qui forment le logiciel et l'hétérogénéité et le nombre important des sites cibles de déploiement. Afin d'automatiser le processus de déploiement de logiciels, il est nécessaire d'avoir un certain nombre d'informations sous la forme d'une description des contraintes de déploiement. Le but de cette description est de fournir des informations suffisantes sur le logiciel pour être en mesure de le déployer d'une manière autonomique. Les plates-formes de déploiement existantes comportent plusieurs formalismes pour exprimer les contraintes de déploiement, les dépendances logicielles et les préférences matérielles des logiciels à déployer. Ces formalismes incluent généralement les langages de description d'architecture (ADL), les descripteurs XML (D& C et CORBA) et les langages dédiés (DSL).

Notre approche consiste à l'utilisation d'un langage dédié pour la description des contraintes de déploiement. Cette approche est similaire à l'approche utiliser dans [Dearle et al., 2004] [Dearle et al., 2010] discuté dans la section 4. L'idée est qu'un administrateur de déploiement écrit les contraintes de déploiement en matière de ressources disponibles en intégrant des mécanismes qui nous offrent beaucoup plus d'expressivité par rapport au langage proposé dans [Dearle et al., 2004], notamment pour traiter les aspects imprévisibles de la topologie (aspects non traités dans [Dearle et al., 2010]). Par exemple, nous voulons exprimer qu'un composant doit être déployé sur tous les hôtes disponibles dans le réseau. Pour cela, nous avons développé j-ASD DSL, un langage de description de contraintes déploiement. j-ASD DSL est un langage déclaratif doté d'une grammaire simplifiée et intuitive réaliser sous la forme d'un plugin Eclipse en utilisant Xtext [20]. Grâce a j-ASD DSL, l'administrateur de déploiement est en mesure de décrire le logiciel à déployer ainsi que les contraintes de déploiement.
Un logiciel est défini par son nom (identifiant), sa version, son URL et les composants qui le forment. Le logiciel est défini aussi par ses dépendances logicielles, ses préférences matérielles et ses contraintes de déploiement. Les types de données supportées par j-ASD DSL sont : chaîne de caractères (String) et entier (Integer). Un logiciel peut être composé par un ou plusieurs composants, chaque composant est défini par le nom du composant, la version du composant, la localisation de son implémentation (URL du composant) et les

20. http://www.eclipse.org/Xtext

dépendances logicielles du composant.
Les contraintes matérielles et logicielles sont, les contraintes sur le système d'exploitation `OsPref`, les contraintes sur le processeur `CPUPref`, les contraintes sur la mémoire disponible `RAMPref`, les contraintes d'affichage `HDPref`, et les contraintes sur la vitesse du réseau `NetSpeedPref`. Cette liste n'est pas exhaustive et peut-être étendue par la suite par d'autres types de contraintes comme la contrainte sur l'utilisation de la batterie `PowerPref` par exemple.

décrit le logiciel à déployer, les composants qui forment le logiciel, les services fournis et requis du logiciel, les dépendances du logiciel, les préférences matérielles et les contraintes de déploiement.

Le programme montré ci-dessous (Fig. 2.16) décrit un logiciel nommé ExtractFromScenario_1, il comporte deux composants (ramSize et display). Les composant ramSize et display sont définis par leurs noms, leurs versions et leurs URL respectives. Les composants ramSize et display sont récupérés à partir d'un serveur http.

```
Software {
  Name=ExtractFromScenario_1
  Version=1
  Components=ramSize display
}
Component {
  Name=ramSize
  Version=1
  Url="http://x.fr/RAM-Size.jar"
}
Component {
  Name=display
  Version=1
  Url="http://x.fr/Display.jar"
}
```

```
HostConstraint {
  Name=Display-Constraint
  CPULoad < 80%
  RAM >= 40 MB
  OSNameContains "Linux"
}
Deployment {
  ramSize @ all
  display @ 1 with Display-
  Constraint
}
```

Figure 2.16 – Exemple de programme écrit avec le DSL j-ASD

La partie hostConstraint représente une spécification des contraintes d'affichage (Display-Constraint) du composant display dans les sites cibles de déploiement. Les contraintes exprimées signifient que dans les sites cibles :

— La charge du processeur (CPULoad) doit être inférieur à 80%.
— Le composant à besoin d'un minimum de 40 Mo de mémoire.
— Le système d'exploitation doit être un système Linux.

Finalement, les contraintes de déploiement sont une spécification de contraintes de haut niveau, qui expriment que le composant ramSize doit être déployé dans tous les sites disponibles au moment du déploiement et le composant display doit être déployé dans un seul hôte qui satisfait les contraintes d'affichage (Display-Constraint).

2.2.2.2 Service de découverte du réseau

Nous proposons un service de découverte du réseau qui permet de détecter dynamiquement l'ensemble des sites disponibles pour servir de cibles de déploiement. Ce service permet à la fois la découverte du réseau local dans le cas où l'utilisateur souhaite déployer son logiciel sur l'ensemble (ou un sous-ensemble) des sites disponibles connectés au réseau lo-

cal ; une découverte étendue (à grande échelle) dans le cas où l'utilisateur souhaite déployer une application à grande échelle ; ou une découverte multi-échelle (découverte mixte du réseau local et à grande échelle).

Pour la réalisation de notre système de découverte nous avons choisi de combiner les protocoles UPnP [Forum, 2008] et XMPP [Saint-Andre et al., 2009]. Le protocole UPnP est utilisé pour la découverte du réseau local, tandis que le protocole XMPP est utilisé pour la découverte à grande échelle. L'idée est que le système de déploiement (le bootstrap), installe un Device et un point de contrôle UPnP sur chaque site cible de déploiement et un Device et un point de contrôle global dans le site initiateur de déploiement dans le cas d'une découverte d'un réseau local. Le point de contrôle global, permet la détection d'une manière totalement transparente les connexions/déconnexions des Device UPnP client dans le réseau et fourni à notre système de déploiement un ensemble d'informations (nom de la machine, adresse IP) qui permettant par la suite le calcule d'un plan de déploiement initial. Dans le cas de la découverte à grande échelle le système de déploiement installe et exécute un client XMPP sur chaque site cible de déploiement et un client spécialisé (client central) dans le site initiateur de déploiement. Tous les clients XMPP seront connectés d'une manière automatique à un serveur XMPP. Une fois qu'un client est connecté au serveur, il envoie d'une manière périodique le couple d'information (nom de machine, adresse IP) au client central. Le client central reste à l'écoute des autres clients et joue le rôle d'un collecteur d'informations provenant des autres clients en exploitons les fonctionnalités de gestion de présence offertes par le protocole XMPP. Ces deux modules de découverte du réseau permet de détecter toutes les situations de nouvelles connexions, de pannes des machines et de déconnexions. Cela permet à notre middleware de déploiement de réaliser un déploiement de logiciel même dans les environnements les plus instables comme les systèmes P2P et ubiquitaire. De plus, la vision globale sur l'état des sites cibles de déploiement ainsi que la détection de défaillances ou des déconnexions représente un atout qui permettra à notre système de déploiement de réaliser des opérations de reconfigurations dynamiques au moment de l'exécution.

2.2.2.3 Bootstrap

Comme nous l'avons vu précédemment, notre système de déploiement doit traiter les problèmes d'administrations multiples et des droits d'accès aux sites cible de déploiement. Nous ne voulons pas contourner les principes de sécurités des systèmes distribués, par conséquent nous comptons que sur les administrateurs des sites pour obtenir les droits d'installation et d'exécution de notre environnement de déploiement dans leurs hôtes. Cela pourrait être réalisé grâce à un programme dédié (le bootstrap) installé volontairement par l'administrateur du site et mis à sa disposition par d'autres moyens. Par exemple, via Bluetooth, ou en envoyant son URL par e-mail ou SMS, ou même en l'intégrant dans un code QR®. Ce programme très léger est un script qui demande aux administrateurs des sites cibles de déploiement (détectés préalablement par le service de découverte du réseau) les droits d'accès (permissions) à l'hôte et met en place l'environnement d'exécution et les dépendances logicielles pour notre middleware. Il contient en particulier les éléments clients du système de découverte du réseau décrit précédemment.

2.2.2.4 Solveur de contraintes de déploiement

Une fois que la description des contraintes de déploiement est fournit sous la forme d'un programme j-ASD DSL, le service de découverte du réseau est lancé pour détecter les sites cibles de déploiement. Ce service retourne une liste de sites cibles potentiels, qui sera la liste initiale des sites cibles de déploiement. Cette liste est passée en entrée du programme j-ASD DSL, ce qui permet de générer un problème de satisfaction de contraintes résolvable par un solveur de contraintes. Un problème de satisfaction de contraintes (CSP) est exprimé par la déclaration d'un ensemble de variables (aux sens inconnus) dont les valeurs sont tirées à partir d'un domaine de valeurs et un ensemble de contraintes sur les variables. Les contraintes sont simplement des relations logiques entre plusieurs variables. La résolution d'un CSP revient à trouver un ensemble consistent de valeurs pour les variables et qui satisfassent toutes les contraintes sur les variables. Nous avons choisi Choco [C.H.O.C.O. Team, 2010] pour notre prototype.

La transformation des contraintes déclarées dans le programme j-ASD DSL est nécessaire à cause de l'écart important entre le niveau d'abstraction dans les programmes de satisfaction de contraintes et les abstractions utilisées par l'administrateur de déploiement pour exprimer les contraintes de déploiement sous la forme d'un programme j-ASD DSL.

Dans le cadre de j-ASD, le problème de satisfaction de contraintes généré par le compilateur est construit à partir d'un ensemble de variables entières (variables de localisation) et un ensemble de contraintes sur ces variables. Nous modélisons le programme CSP avec les éléments suivants :
Un ensemble fini C de composants logiciels. Par exemple, dans l'exemple présenté précédemment (Fig. 2.16) l'ensemble C = {display, ramSize}.
Un ensemble de sites cibles de déploiement H détectés par le service de découverte du réseau.
Un ensemble de variables de localisation (Loc), qui modélisent la localisation d'un composant sur un site tel que :
Loc (Ci, Hj) = 1, si le composant Ci peut être installé dans le site Hj.
Loc (Ci, Hj) = 0, si le composant Ci ne peut pas être installé dans le site Hj.
Un ensemble P de contraintes sur les sites cible, par exemple le niveau de charge du processeur et la taille de la mémoire disponible.
Un ensemble de contraintes sur les variables de localisation (Loc (Ci, Hj)).
Un problème de placement initial de composants sur l'ensemble des hôtes H en respectant les contraintes de déploiement.

Les contraintes de déploiement déclarées dans le programme j-ASD DSL seront traduites par le compilateur en des contraintes sur les variables de localisation (loc (Ci, Hj)) comme montré dans l'exemple ci-dessous (Fig. 2.17).

La première contrainte signifie que le composant C1 doit être déployé dans tous les sites cibles de déploiement, cela signifie formellement :
$c1 \in C, \forall Hj \in H$, loc (c1, Hj) = 1.
La deuxième contrainte signifie que le composant c2 doit être déployé dans un sous-ensemble de trois sites. La troisième contrainte signifie que le composant c3 doit être déployé dans tous les sites qui satisfassent les contraintes définies dans Constr1. Autrement dit, le composant c3, doit être installé dans tous les sites qui comportent au minimum 150 Mo de mémoire disponible et équipés d'un système d'exploitation de type Windows.

Formellement :

```
HostConstraint {
  Name=Constr1
  RAM >= 150 MB
  OSNameContains "Windows"
}
HostConstraint {
  Name=Constr2
  CPULoad < 60%
  RAM >= 100 MB
  OSNameContains "Linux"
}
```

```
Deployment {
c1 @ all;
c2 @ 3;
c3 @ ALL with Constr1;
c4 @ 4 hosts with Constr2;
}
```


Figure 2.17 – Second exemple de programme écrit avec le DSL j-ASD

$c3 \in C, \forall Hj \in H$, **si** $((RAM \geqslant 150MB)$ et $(OSName = "Windows"))$ **alors** loc (c3, Hj) = 1 **sinon** loc (c3, Hj) = 0.
La quatrième contrainte signifie que le composant c4 doit être installé dans un sous-ensemble de quatre sites qui respectent l'ensemble de contraintes définies dans Constr2.

Le solveur retourne la première solution valide du CSP si il en existe une. Elle se présente sous la forme d'un ensemble de variables, qui sera directement interprêtée comme un plan de déploiement initial par le système d'agents mobiles. Si il n'y a pas de solution, un message d'erreur est renvoyé à l'utilisateur qui devra revoir ses contraintes pour réussir un déploiement. Il est possible que le solveur de contraintes arrive à trouver plusieurs solutions consistantes et différentes (plusieurs plans de déploiement). Néanmoins, lorsque certains paramètres de qualité de service sont considérés, certaines de ces solutions sont meilleures que d'autres. Notre objectif n'est pas de trouver la meilleure solution possible, pour l'instant notre objectif est de trouver une première solution consistante qui satisfasse les contraintes de déploiement. Le travail sur la sélection de la meilleure solution est une perspective intéressante de ce travail.

2.2.2.5 Support de déploiement

Le support de déploiement fourni l'environnement d'exécution pour notre système de déploiement. Il doit permettre aussi l'installation, la désinstallation, l'activation, la désactivation et la mise à jour des composants déployés au moment de l'exécution sans redémarrer l'ensemble du système. Le système de déploiement doit également permettre le déploiement de logiciel sur des hôtes hétérogènes et à ressources réduites tel que les ordinateurs portables, les tablettes tactiles, les Smartphones, les voitures et les PC-Ultra mobile. Nous ne visons pas dans ce travail des environnements plus petits que ceux décrits.

Plusieurs plates-formes de déploiement existantes comme OSGi et D&C fournissent tout ou une partie des fonctionnalités souhaitées. Pour la réalisation de notre prototype, nous avons choisi la plate-forme OSGi[21] comme support de déploiement. Les motivations de notre choix de plate-forme et d'unité de déploiement sont discutées dans la section suivante.

21. http://www.osgi.org

2.2.2.6 Système d'agents mobiles

L'utilisation des agents mobiles pour l'automatisation du processus de déploiement n'est pas une nouvelle approche. Il existe plusieurs travaux dans la littérature qui ont utilisé cette technique pour déployer des logiciels dans des infrastructures réseau statique (voir par exemple [Hall et al., 1999]). Généralement, les solutions proposées ne sont pas adaptées au déploiement de logiciel dans des environnements ouverts tels que les systèmes ubiquitaires.

Un agent logiciel peut être défini comme un programme autonome caractérisé par des données et un comportement privé [Bradshaw, 1997]. Un agent mobile est un logiciel capable de se déplacer au moment de l'exécution avec son code et ses données [Harrison et al., 1995]. Un agent mobile adaptable (AMA) [Leriche and Arcangeli, 2010] est une entité logicielle autonome capable de communiquer et de se déplacer, disposant d'un comportement privé et d'une capacité d'adaptation au contexte de l'exécution. Les agents logiciels communiquent généralement par envois de message en mode asynchrone. Un système d'agents mobiles est un Framework qui implémente le paradigme des agents mobiles [Filho et al., 2000], il fournit des primitives et des services qui permettent l'implémentation, les communications et la migration des agents logiciels.

Nous utilisons un système d'agents mobiles adaptables pour exécuter et superviser le processus de déploiement. Pour cela, nous avons créé des agents de supervision (global et local) et des agents de déploiement. Les agents de supervision ont pour rôle l'exécution, la supervision, le contrôle et la reconfiguration du processus de déploiement. Les opérations de reconfiguration et d'adaptation sont exécutées pour réagir aux changements rencontrés dans l'environnement d'exécution dans lequel le logiciel est déployé (pannes de machine ou de liens, déconnexion, par exemple). Afin d'assurer le passage à l'échelle nous proposons deux types d'agents de supervisions, des agents de supervision locaux (LSA) et un agent de supervision global (GSA).

Un agent de supervision local est déployé par l'agent de supervision global sur un site cible d'un sous réseau local (un sous réseau de classe C pour des raisons de passage à l'échelle). L'agent de supervision local a pour rôle la création des agents de déploiement dans chaque site cible du sous-réseau pour installer, activer, désactiver et mettre à jour le logiciel à la demande de l'agent de supervision global. Il a aussi pour rôle la supervision du processus de déploiement et la réalisation des opérations de reconfiguration dynamique dans le sous-réseau.

L'agent de supervision global est créé au début du processus de déploiement. Il a pour rôle l'exécution et la supervision du plan de déploiement initial calculer par le solveur de contraintes. Il n'y a qu'un seul agent de supervision global dans notre système créé et exécuté initialement dans le site initiateur du processus de déploiement. Il contrôle le processus de déploiement par la coordination avec les agents de supervision locaux. Pour cela, l'agent de supervision local échange des messages asynchrones avec les LSA pour connaître l'état des activités de déploiement dans chaque sous-réseau. L'agent de supervision global fourni aussi des interfaces d'échange à l'administrateur de déploiement qui lui permet de vérifier et d'intervenir à tout moment dans l'une des activités du processus de déploiement. Cet agent peut aussi prendre d'une manière autonome des décisions de migrations vers d'autres sites pour réagir aux variations de qualité de service et aux pannes de machines et de déconnexion.

L'agent de déploiement est chargé d'exécuter les activités de déploiement dans

chaque site cible de déploiement. Il est créé et exécuté dans un site cible selon le plan de déploiement. L'agent de déploiement réalise plusieurs opérations telles que, le téléchargement des composants (à partir d'un serveur web par exemple), la résolution des dépendances logicielles, l'installation des composants dans le site cible et la notification de la fin de l'activité d'installation à l'agent de supervision par un message asynchrone. L'agent de déploiement permet aussi l'activation, la désactivation et la désinstallation des composants de l'application à déployer à la demande de l'administrateur de déploiement ou l'agent de supervision.

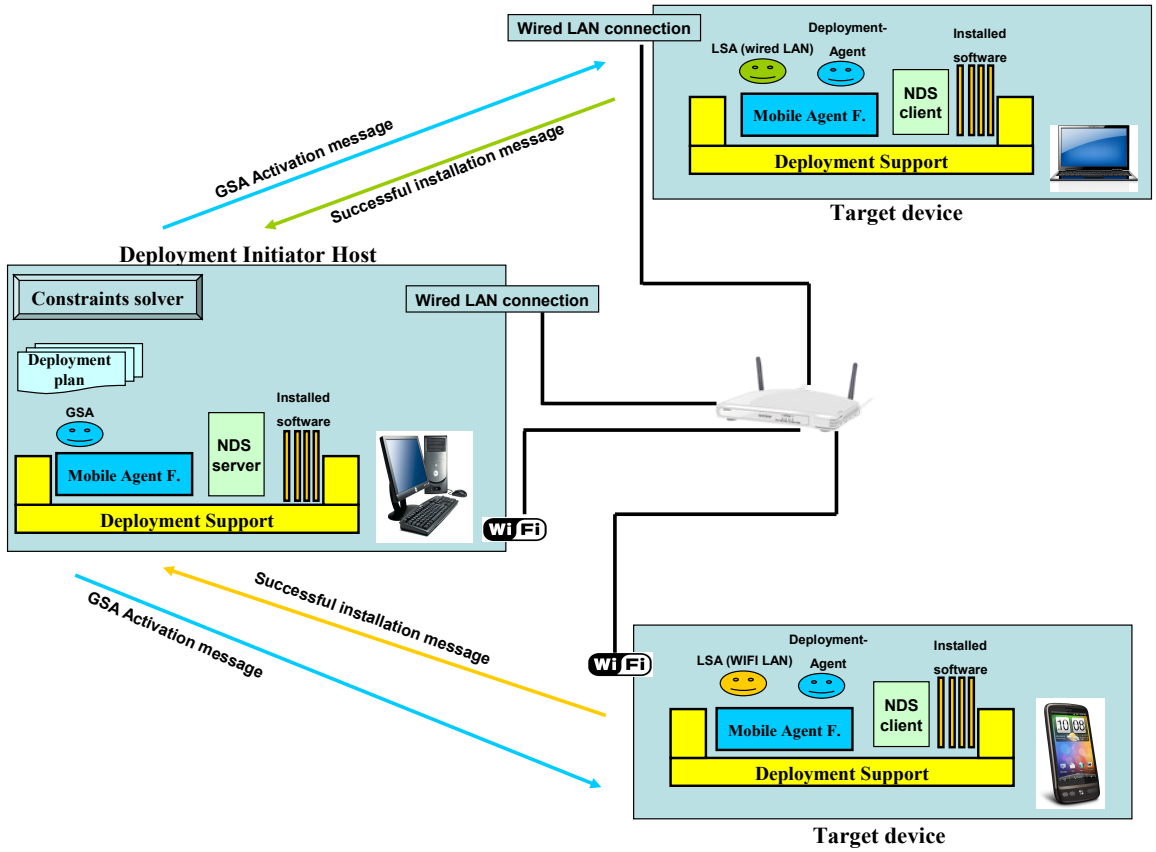

Figure 2.18 – Agents de supervision

Après l'installation d'un (ou plusieurs) composant, l'agent de déploiement effectue une opération de vérification locale de l'échec/succès de l'activité d'installation et envoie un message de notification à l'agent de supervision local selon la situation (message de succès ou d'échec de l'installation). Une fois le message est reçu, l'agent de supervision local traite localement les messages reçus et notifie par la suite l'information de l'échec/succès de l'installation au GSA par l'envoi d'un message de succès ou d'échec d'installation dans le sous-réseau.
Le GSA envoie un message d'activation à tous les agents de supervision locaux une fois qu'il a reçu tous messages de succès de l'installation comme le montre la figure 2.18. Le comportement des agents de supervision locaux sera la création et l'envoi des messages d'activation à tous les agents de déploiement déployés dans le sous-réseau. Si ls GSA ne reçoit aucun message (succès/échec de l'installation par exemple) de la part d'un agent de supervision local (à cause d'une déconnexion ou panne de machine par exemple) dans un intervalle de temps

fini, il procède à une procédure de reconfiguration, qui consiste à la désactivation de tous les agents de supervision locaux et l'envoi d'un message d'auto destruction (suicide) à l'agent qui ne répond pas. Ce dernier, une fois qu'il a reçu le message de suicide, envoie un message d'auto destruction aux agents de déploiement créés et termine sa propre exécution. En parallèle l'agent de supervision global créé un nouveau LSA dans un autre site du sous-réseau concerné en respectant les contraintes de déploiement et envoie un message d'activation à tous les LSAs désactivés après la réception du message de succès de l'installation de la part du nouveau LSA.

Il est à noter que l'agent de supervision local peut migrer vers un autre site cible de déploiement dans le sous-réseau sans consulter l'agent de supervision global afin de corriger une défaillance détectée (l'agent dispose de moins de mémoire par exemple). Cela offre à notre middleware la possibilité de réaliser des opérations de reconfiguration dynamique et autonomique au moment de l'exécution. Le LSA informe par la suite l'agent de supervision global de ça nouvelle position après la correction de la situation.supervision local ne peut pas résoudre localement une défaillance, le GSA sera informé afin de trouver une solution. Le GSA peut par exemple recalculer un nouveau plan de déploiement afin de trouver une solution à cette défaillance. Dans le cas où aucune solution n'est trouvée, le GSA mettra fin au processus de déploiement et demande à l'administrateur de déploiement de réduire l'ensemble des contraintes proposées.

2.2.2.7 Synthèse

Dans cette section nous avons présenté notre middleware j-ASD dédié au déploiement autonomique de logiciels en environnement ubiquitaire ainsi que certains éléments du prototype qui permet la validation de notre approche. La solution proposée est composée d'un langage dédié (DSL) à la description des contraintes de déploiement, un service réseau pour découvrir d'une manière automatique les sites cibles de déploiement, un logiciel de bootstrap pour la préparation de l'environnement d'exécution, un solveur de contraintes pour le calcul d'un plan déploiement initial et un système d'agents mobiles adaptables pour l'exécution et la supervision des activités de déploiement. Nous estimons que j-ASD peut répondre aux exigences et aux spécificités des environnements répartis ouverts dotés d'une topologie dynamique du réseau. Nous avons utilisé le Framework OSGi comme support de déploiement pour permettre le déploiement d'applications basées sur des bundles OSGi sur plusieurs types d'équipement hétérogènes (PDA, téléphones mobiles, Smartphones, tablettes et ordinateur). Le service de découverte de réseau et le logiciel de bootstrap permettent la détection et la gestion des droits d'accès aux sites cibles de déploiement avec un minimum d'intervention humaine. La solution générée par le solveur de contraintes est dynamiquement interprêtée comme plan de déploiement initial par le système d'agents mobiles. Les caractéristiques des agents mobiles (comportement autonome et capacité de migration) nous permettent d'effectuer des adaptations et des reconfigurations dynamiques au moment de l'exécution sans aucune intervention de l'administrateur de déploiement.

Nous poursuivons actuellement nos travaux sur les tests et les évaluations de la dernière version du prototype j-ASD sur des environnements répartis à grande échelle. En parallèle nous expérimentons des algorithmes de déploiement plus intelligents pour faire face aux besoins d'adaptations dans des situations plus complexes après la réalisation du déploiement initial.

2.2.3 MuScADeP : un processus pour le déploiement autonome

La norme [ISO 9000 :2005, 2009] définit un processus comme un ensemble d'activités corrélées ou interactives qui transforme des éléments d'entrée en éléments de sortie. La conception, puis la réalisation et éventuellement le contrôle du déploiement sont les activités principales du processus de déploiement qui prend en entrée les éléments à déployer et leurs caractéristiques, et permet en sortie d'obtenir une application disponible à l'utilisation, selon la définition de Carzaniga *et al*.

Les processus de déploiement usuels sont centralisés et dirigés par un opérateur humain. Ils sont adaptés au déploiement d'applications dans un environnement hiérarchique, tels un réseau d'entreprise ou une grappe de serveurs. Ils sont généralement pilotés depuis une unité centrale où un opérateur humain cumule les rôles de concepteur et d'opérateur du déploiement.

Nous avons montré qu'une solution de déploiement pour les systèmes répartis multi-échelles devait organiser et automatiser un certain nombre d'opérations dans le cadre d'un processus en partie décentralisé et sensible au contexte. Ce processus doit permettre la prise en compte de la dynamique du domaine de déploiement ainsi que celle du système logiciel, et permettre une séparation claire entre les rôles. En particulier, celui de concepteur doit être dévolu à un opérateur humain alors que le rôle d'opérateur peut être pris par le système de déploiement. Pour cela, il nous semble pertinent de repenser et définir un processus de déploiement adéquat.

2.2.3.1 Vue globale du processus

Le processus que nous proposons organise les différentes activités de déploiement de la manière suivante. Dans un premier temps, le concepteur exprime une spécification. Puis, un plan de déploiement initial de l'application est généré automatiquement en fonction de la spécification et de l'état initial du domaine. Ensuite, le plan de déploiement initial est réalisé, puis révisé à l'exécution afin d'être adapté à la dynamique de l'application et à celle du domaine, ceci pour maintenir l'application en condition opérationnelle : cette partie du processus est appelée « déploiement dynamique ».

Dans ce processus, le rôle d'opérateur du déploiement est joué par une application de niveau middleware : le système de déploiement. C'est ce système de déploiement automatique (et autonomique) qui réalise les différentes opérations de déploiement.

Cette approche suppose une acquisition de l'état du domaine, initialement pour produire les données nécessaires à la génération du plan en fonction des appareils présents et des ressources disponibles, puis dynamiquement pendant l'exécution de l'application.

Pour décrire le processus, nous utilisons une notation graphique inspirée de SPEM [SPEM2.0, 2008]. La légende des diagrammes est donnée dans la figure 2.19. Trois types d'actions seront décrites (différenciées par leurs couleurs) : (i) les actions centralisées, qui sont pilotées par une unité centrale, et s'effectuent sur un seul appareil, (ii) les actions mixtes, pilotées par une unité centrale mais dont l'action est réalisée sur l'ensemble ou une partie du domaine de déploiement, (iii) les actions décentralisées qui sont effectuées sur l'ensemble ou une partie du domaine de déploiement, sans initiative centrale. Sauf indication contraire, les actions décentralisées sont initiées par le système de déploiement.

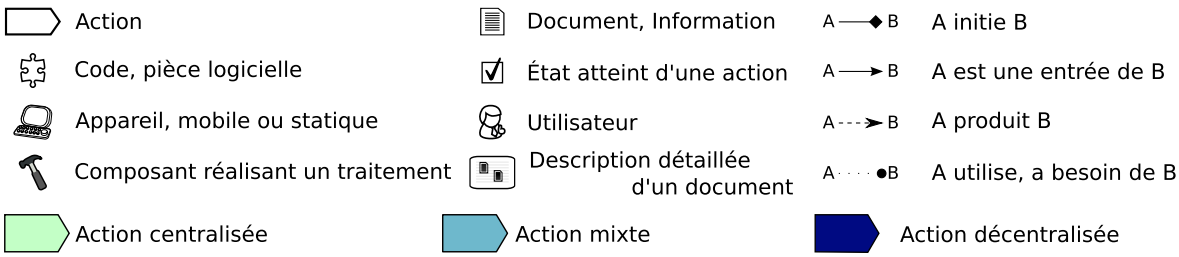

Figure 2.19 – Légende des diagrammes de processus

La figure 2.20 donne une vue globale du processus de déploiement.

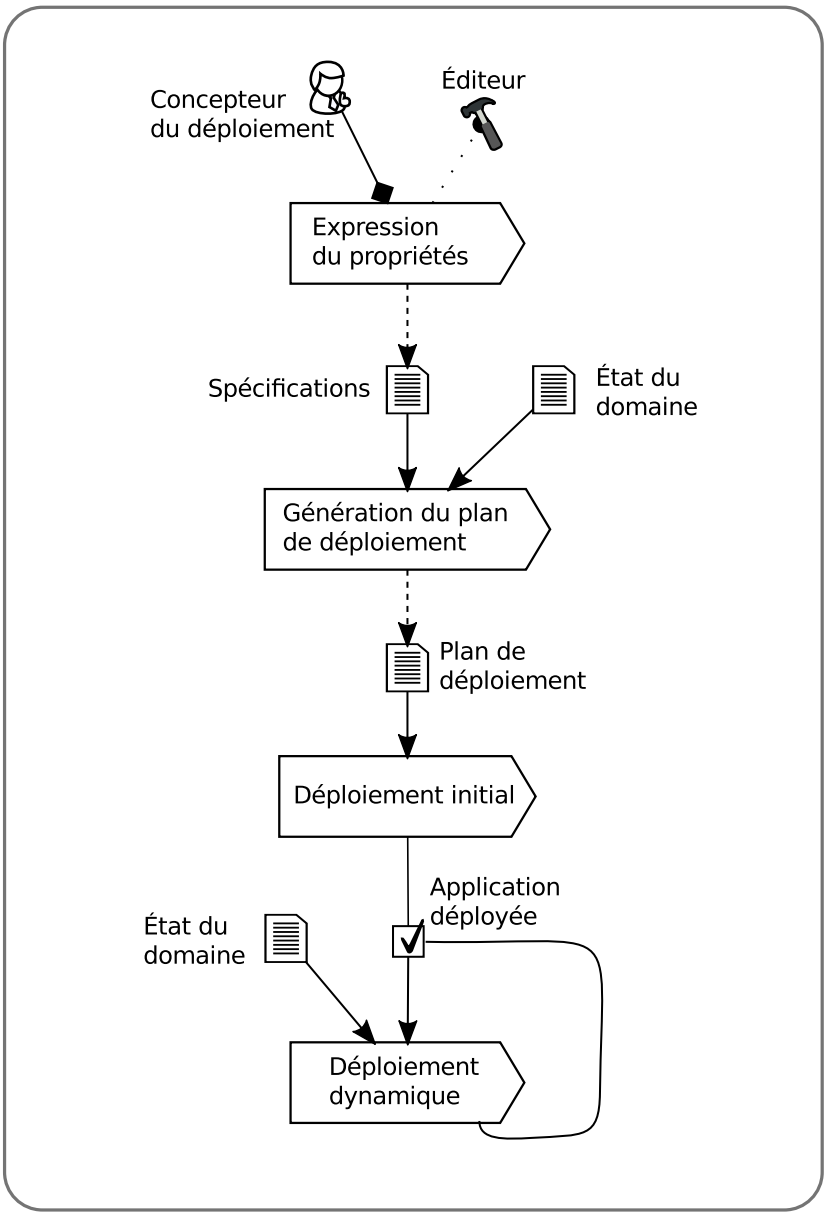

Figure 2.20 – Vue globale du processus de déploiement

Ce processus permet d'organiser les différentes étapes du déploiement. Dans un pre-

mier temps, le concepteur du déploiement spécifie les propriétés de déploiement. Puis, ces spécifications sont données en entrée d'un générateur de plan, avec l'état du domaine récupéré à début de l'exécution du déploiement. Ce plan de déploiement est ensuite réalisé : les composants sont installés et activés. Dans un second temps, le système de déploiement réagit autonomiquement au dynamisme du domaine de déploiement ainsi qu'à celui de l'application. Le déploiement incrémental permet de prendre en compte la dynamique de l'application, alors que le déploiement continu permet celle du domaine. Une boucle de contrôle autonomique permet de prendre en compte les variations de l'état du domaine (dégradation de l'état d'un appareil, disparition, etc.).

2.2.4 MuScADeL : un langage dédié à l'expression du déploiement

Comme nous l'avons présenté plus tôt, dans le cadre des systèmes répartis multi-échelles, le concepteur de déploiement ne peut pas exprimer de plan de déploiement. La dynamique du domaine ne permet pas une connaissance précise des appareils impliqués dans le déploiement au moment de la conception. Le concepteur doit être capable de spécifier le déploiement de son application sans avoir une connaissance exacte du domaine de déploiement. Pour ce faire, un support d'expression offrant un haut niveau d'abstraction doit lui permettre de spécifier les différentes propriétés de son application (code, contraintes, versions, dépendances entre composants, etc.).

Le déploiement des systèmes répartis multi-échelles s'effectue en deux étapes, le déploiement initial et le déploiement dynamique (cf. section 2.2.3.1). Un langage doit supporter l'expression de propriétés relatives à chacune des étapes : l'expression de propriétés à vérifier initialement, et l'expression des propriétés à vérifier à l'exécution du système. Afin de pouvoir générer le plan de déploiement initial, le support d'expression doit aussi permettre au concepteur d'exprimer des informations quant au domaine de déploiement : les informations utiles à la récupération des informations matérielles et logicielles spécifiques aux propriétés exprimées. De plus, le support d'expression doit aussi permettre la spécification d'informations relatives à la dynamique de l'application et la dynamique du domaine de déploiement. Le système déployé étant multi-échelle, le support d'expression doit permettre au concepteur d'exprimer des propriétés relatives aux aspects multi-échelles, spécifiques à son système.

La conception du déploiement requiert des compétences et des moyens particuliers. Ainsi, c'est un langage dédié (DSL, *Domain Specific Language*)à l'expression des propriétés de déploiement des systèmes répartis multi-échelles que nous proposons, le langage MuScADeL (***Mu**lti**Sc**ale **A**utonomic **De**ployment **L**anguage*). De manière générale, les DSLs présentent différents avantages. Ils utilisent les idiomes et les abstractions du domaine visé, et peuvent ainsi être utilisés par les experts du domaine. Ils sont légers, faciles à maintenir, portables et réutilisables. Ils sont le plus souvent bien documentés, cohérents et fiables, et optimisés pour le domaine visé [Van Deursen et al., 2000, Strembeck and Zdun, 2009, Tolvanen and Kelly, 2010].

La grammaire est définie en utilisant la syntaxe EBNF. Le code est présenté en cinq extraits présentant respectivement les composants (listing 2.1), les critères (listing 2.2), les sondes (listing 2.3), les sondes multi-échelles (listing 2.4) et les exigences de déploiement (listing 2.5).

2.2.4.1 Composant

L'unité de déploiement est le composant. Le descripteur (ou code) MuScADeL liste les types de composants du système, utilisant le mot-clé `Component` (listing 2.1). Une description d'un type de composant comporte plusieurs champs. Le champ `Version` est utile pour l'activité de mise à jour. Le champ `URL` spécifie l'adresse à laquelle le code du composant est téléchargeable. Le champ `DeploymentInterface` spécifie l'interface de contrôle du composant. Il est essentiel pour permettre l'interaction entre le composant et le système de déploiement : ce dernier doit interagir avec le composant pour le configurer, l'activer, l'administrer en cours d'exécution et le désactiver. Le champ `Dependency` liste les composants requis : à l'installation du composant, si un composant qu'il requiert n'est pas déployé, le système de déploiement doit le déployer sur l'appareil. Des contraintes peuvent être ajoutées à un composant. Le champ `Constraint` liste les conditions logicielles et matérielles que le déploiement doit respecter, définies avec le mot-clé `BCriterion`, par exemple à la ligne 2 du listing 2.2. Par défaut, les contraintes doivent être satisfaites à la fois lors de la génération du plan de déploiement et lors de l'exécution du système (le système de déploiement doit vérifier dynamiquement qu'il n'y a pas de violation de contraintes). Des contraintes qui doivent être satisfaites uniquement à l'installation peuvent être définies utilisant le mot-clé `InitOnly`, elles ne seront pas vérifiées à l'exécution du système (ligne 16 du listing 2.1).

```
1  Component Bike {
2      Version 1
3      URL "http://income.fr/4ME/bike.jar"
4  }

6  Component BikeAvail {
7      Version 1
8      URL "http://income.fr/4ME/bikeAvail/jar"
9      DeploymentInterface fr.enac.plop.DIimpl
10 }

12 Component GUI {
13     Version 1
14     URL "http://income.fr/4ME/gui.jar"
15     Constraint RAM80
16     InitOnly Constraint Screen
17 }

19 Component IM {
20     Version 1
21     URL "http://income.fr/4ME/im.jar"
22     Dependency GUI
23     InitOnly Constraint RAM80
24 }

26 Component Stat {
27     Version 2
28     URL "http://income.fr/4ME/stat.jar"
29     Constraint CPURAM
30 }
```

Listing 2.1 – `Component` – Spécification des composants dans MuScADeL

Par exemple, le type de composant `GUI` est en version 1, disponible à l'adresse `http://income.fr/4ME/gui.jar`. À son installation, l'écran de l'appareil doit être d'une taille

d'au moins 4 pouces et lors de l'installation et l'exécution du composant, l'appareil doit disposer d'au moins 80Mo de mémoire vive disponible.

2.2.4.2 Critère

De manière générale, les critères sont des conditions à respecter. Ils sont utilisés pour la spécification de contraintes propres à un composant ou la spécification d'exigences de déploiement. Le mot-clé `BCriterion` définit un critère (ici, il s'agit de critères de base par opposition aux critères multi-échelles, introduits dans la section 2.2.4.4). Un critère est une condition, une conjonction ou une disjonction de conditions portant sur des valeurs sondées, comme `CPURAM` (listing 2.2, ligne 5). Il existe deux types de conditions, l'une concernant l'existence ou l'activité d'une sonde, l'autre concernant la valeur donnée d'une sonde.

Dans le premier cas, la condition est composée du nom de la sonde et des mots-clés `Exists` ou `Active`, dont les méthodes sont définies dans toutes les interfaces des sondes. Par exemple, dans le listing 2.2, à la ligne 2, la sonde utilisée est `SigFox`, et la condition utilise les méthodes définies par défaut `Exists` et `Active`.

Si la condition est une condition valuée, elle est composée du nom de la sonde, de la méthode à appeler, d'un comparateur et de la valeur. Dans ce cas, la méthode est spécifique à la sonde et est définie dans l'interface de la sonde. Par exemple, dans le listing 2.2, à la ligne 7, la sonde utilisée est `RAM`, la méthode utilisée est `free` et la valeur comparée est le nombre `1`, pour 1Mo. Un critère peut être utilisé pour définir une contrainte portant sur un composant (listing 2.1, ligne 15) ou une exigence de déploiement (listing 2.5, ligne 7). Le critère `LinuxOrAndroid` (ligne 10) utilise l'opérateur `|` de disjonction entre deux conditions. Ce critère défini que l'appareil doit avoir comme système d'exploitation Linux ou Android. La disjonction des conditions est prioritaire à la conjonction des conditions.

```
1  BCriterion SigFoxActive {
2    SigFox Exists; SigFox Active;
3  }

5  BCriterion CPURAM {
6    CPU.proc > 1;  // at least 1Ghz of CPU
7    RAM.free > 1;  // at least 1Mb free RAM
8  }

10 BCriterion LinuxOrAndroid {
11     OS.name = "Linux"
12     | OS.name = "Android";
13 }
```

Listing 2.2 – `BCriterion` – Spécification des critères dans MuScADeL

2.2.4.3 Sonde

Les sondes logicielles sont utilisées dans la définition des conditions des critères. Elles permettent de récupérer des informations précises concernant l'appareil hôte. Les sondes logicielles sont définies en utilisant mot-clé `Probe`. Elle comporte deux champs obligatoires. Le premier champ est `ProbeInterface`, qui spécifie l'interface de la sonde. Le second est le champ `URL`, qui indique l'adresse de téléchargement du code de la sonde.

```
1  Probe SigFox {
```

```
2       ProbeInterface fr.irit.sigfox.DIImpl
3       URL "http://irit.fr/INCOME/SigFoxProbe.jar"
4   }
```

Listing 2.3 – `Probe` – Spécification des sondes dans MuScADeL

Sonde multi-échelle La conception du système réparti multi-échelle est basée sur une caractérisation multi-échelle du système. Cette caractérisation définit les points de vue, les dimensions et les échelles de l'application. Les sondes multi-échelles permettent de récupérer des informations multi-échelles sur l'appareil hôte. Les sondes multi-échelles sont définies en utilisant le mot-clé `MultiScaleProbe`. Comme `Probe`, elle n'a que deux champs, `MultiScaleProbeInterface` et `URL`, qui spécifient respectivement l'interface de la sonde multi-échelle et l'adresse à laquelle son code est téléchargeable.

Une sonde est définie par point de vue. Les sondes permettent d'identifier la dimension, l'échelle et l'instance d'échelle d'un appareil donné. L'instance d'une échelle est la valeur spécifique à l'appareil de l'échelle à laquelle il appartient. Un mot-clé spécifique est nécessaire aux sondes multi-échelles car elles ont un traitement opérationnel spécifique lors de la récupération de l'état du domaine. Le code MuScADeL des sondes multi-échelles peut être généré par le framework MuSCa.

```
1   MultiScaleProbe Device {
2     MultiScaleProbeInterface eu.telecom-sudparis.DeviceProbeImpl
3     URL "http://it-sudparis.eu/INCOME/DeviceProbe.jar"
4   }
5   MultiScaleProbe Administration {
6     MultiScaleProbeInterface eu.telecom-sudparis.AdminProbeImpl
7     URL "http://it-sudparis.eu/INCOME/AdminProbe.jar"
8   }
```

Listing 2.4 – `MultiScaleProbe` – Spécification des sondes multi-échelles dans MuScADeL

2.2.4.4 Déploiement multi-échelle

Une fois tous ces éléments définis, les exigences de déploiement du système multi-échelle peuvent être spécifiées. L'opérateur `@` permet de spécifier une conjonction d'exigences spécifiques à un composant. L'ensemble des exigences peuvent prendre plusieurs formes, tel que présenté dans le listing 2.5.

Le mot-clé `AllHosts` permet de circonscrire le domaine de déploiement : la ligne 2 permet de définir que le déploiement doit s'effectuer sur tous les appareils satisfaisant le critère `LinuxOrAndroid`.

```
1   Deployment {
2       AllHosts LinuxOrAndroid;
3
4       Bike @ All, Administration.Level.Service("Toulouse.SharingBikes"),
5               Device.Type.Cloudlet;
6       BikeAvail @ 1/5 Bike, SigFoxActive;
7       Stat @ 10..30, Device.Type.Cloudlet;
8       GUI @ All, Device.Types.Smartphone;
9       IM @ All, Device.Type.Smartphone,
10                  User.NumberOfUsers.Group;
11      IMHist @ Device.Type.Smartphone,
12                      Each User.NumberOfUsers.Group;
```

```
13      RoutePlanner @ 2, SameValue Administration.level.service(Bike),
14                        Device.StorageCapacity.Giga;
15  }
```

Listing 2.5 – `Deployment` – Spécification des exigences de déploiement dans MuScADeL

Le concepteur doit néanmoins pouvoir identifier de désigner des parties du domaine, sans qu'il ne puisse désigner explicitement la totalité des appareils. Ce sont les notions d'échelle et d'instance d'échelle qui apportent une solution à ce problème et permettent de spécifier le déploiement d'un composant dans telle ville, sur un tel réseau local, sur des appareils appartenant à tel domaine administratif, etc. Le listing 2.5 montre un certain nombre d'exigences de déploiement à une certaine échelle :

- Le composant `Bike` doit être déployé sur tous les appareils qui font partie de *(i)* l'échelle `Cloudlet` dans la dimension `Type` du point de vue `Device`, et *(ii)* sont administrés par le service `"Toulouse.SharingBikes"` — instance `"Toulouse.SharingBikes"` dans l'échelle `Administration.Level.Service` — (lignes 4 et 5) ;
- Les composants `BikeAvail` doivent être déployés sur des appareils satisfaisant le critère `SigFoxActive`, l'expression du ratio `1/5` permettant de spécifier qu'un composant `BikeAvail` doit être déployé pour chaque lot de cinq composants `Bike` déployés (ligne 6) ;
- Entre dix et trente composants `Stat` doivent être déployés sur des appareils appartenant à l'échelle `Device.Type.Cloudlet` (ligne 7) ;
- Le composant `GUI` doit être déployé sur tous les appareils qui appartiennent à l'échelle `Device.Type.Smartphone`, c'est-à-dire sur les smartphones du domaine de déploiement, y compris ceux entrant dans le domaine de déploiement à l'exécution du système, utilisant le mot-clé `All` (ligne 8) ;
- Le composant `IMHist` doit être déployé sur un smartphone dans chaque instance de l'échelle `User.NumberOfUsers.Group` (mot-clé `Each`), c'est-à-dire, un smartphone de chaque groupe de personnes (lignes 11 et 12) ;
- Le composant `RoutePlanner` doit être déployé sur deux appareils *(i)* ayant une grande puissance de calcul (appartenant à l'échelle `Device.StorageCapacity.Giga`), et *(ii)* étant administré par le même service d'entreprise que le composant `Bike`, en utilisant le mot-clé `SameValue` (lignes 13 et 14).

Le descripteur MuScADeL doit contenir au moins une définition d'un composant et l'expression d'un déploiement. Les autres sections sont optionnelles. Le code peut être découpé en plusieurs fichiers, le mot-clé `Include` permet d'inclure d'autres fichiers.

2.2.4.5 MuScADeL et la gestion de la dynamique

Certaines constructions du DSL sont adaptées à l'expression de propriétés relatives à la dynamique et à l'ouverture. Par défaut, les contraintes doivent être satisfaites initialement, dès la génération du plan de déploiement, puis durant l'exécution de l'application. Elles doivent donc être vérifiées dynamiquement. Le mot-clé `InitOnly` permet de spécifier qu'une (ou plusieurs) contraintes doivent être satisfaites initialement mais non nécessairement à l'exécution.

D'autre part, lorsque le déploiement est spécifié, dans la section `Deployment`, le mot-clé `All` permet d'exprimer qu'un composant doit être déployé sur un sous-domaine qui

satisfait dynamiquement la propriété exprimée. Dans l'exemple, le composant `GUI` doit être déployé sur tous les smartphones du domaine, incluant ceux qui entrent dans le domaine alors que l'application s'exécute. Il en est de même pour la propriété de ratio. Ainsi, le plan de déploiement évolue dynamiquement en fonction des entrées et des sorties des appareils composant le domaine de déploiement.

2.2.4.6 Conclusion

MuScADeL est notre proposition de langage dédié au déploiement de systèmes répartis multi-échelles. L'utilisation d'un DSL permet de fournir un moyen d'expression du déploiement avec haut niveau d'abstraction. MuScADeL permet au concepteur non expert en déploiement (mais expert dans le domaine de l'application à déployer) de spécifier les composants de l'application et leurs relations, ainsi que les propriétés de déploiement. Ces propriétés sont des contraintes d'exécution propres aux composants et des exigences de déploiement (qui sont les choix du concepteur). Certaines expressions du langage sont spécifiques à la dynamique du domaine de déploiement. Ainsi, le concepteur peut directement donner des directives pour la prise en compte du déploiement continu de l'application à déployer. De plus, MuScADeL permet la spécification d'exigences en fonction des caractéristiques multi-échelles du système à déployer, ainsi que la définition de sondes et de critères multi-échelles. De par le couplage entre MuSCa et MuScADeL, ces exigences sont vérifiées et validées par l'éditeur, et afin de simplifier la tâche au concepteur, le code MuScADeL relatif aux sondes multi-échelles est généré par MuSCa.

2.2.5 MuScADeM : l'intergiciel final

Nous présentons ici notre réalisation de la solution pour le déploiement de systèmes répartis multi-échelles : il s'agit de MuScADeL et de l'outillage associé et du middleware MuScADeM. À partir d'un descripteur MuScADeL, MuScADeM permet la récupération de l'état du domaine de déploiement, la génération d'un plan de déploiement, sa réalisation, et le maintien en condition opérationnelle de l'application. MuScADeM contient une partie centralisée, le moniteur de déploiement (là où s'effectue la génération du plan de déploiement), et une partie décentralisée, le support local du système de déploiement sur les appareils.

Le processus de déploiement initial du système réparti multi-échelle présenté dans la figure 2.21, dans laquelle nous avons précisé les outils technologiques utilisés.

Nous suivons l'ordre chronologique de ce processus pour présenter nos réalisations et choix technologiques.

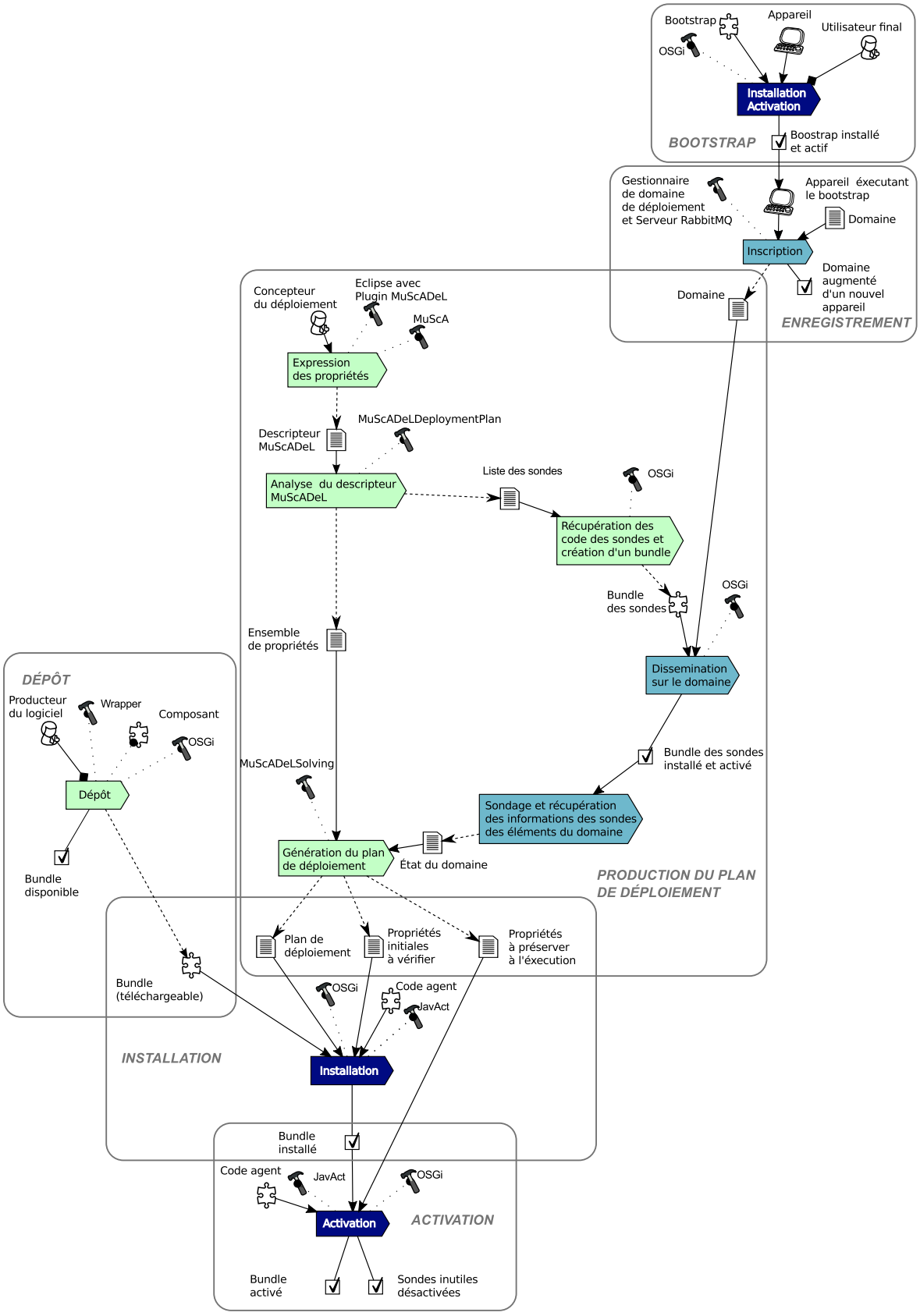

Figure 2.21 – Processus de déploiement initial avec les choix technologiques

2.2.5.1 Bootstrap

Le bootstrap est un système d'amorce installé sur l'appareil, sur lequel s'appuie le système de déploiement. C'est un conteneur OSGI contenant différents bundles de base – le bundle est l'unité de déploiement du framework OSGI. Par la suite, le support local du système de déploiement va s'enrichir en ajoutant des bundles à ce bootstrap. L'implémentation d'OSGi utilisée est Apache Felix [Felix, xx], version 4.2.1. Ce choix a été fait d'une part car cette implémentation est open source, et d'autre part parce qu'elle est facilement déployable sur des dispositifs de petite taille, tels des smartphones sous Android.

En pratique, pour des raisons d'hétérogénéité, il existe deux versions du bootstrap : une pour Java (J2SE standard) et une pour Android (Dalvik). Les différences entre ces deux versions résident dans l'implémentation du code de lancement de Felix, soit destinée à un smartphone Android, soit du J2SE. Les bundles composant le bootstrap sont eux par contre génériques.

Le bootstrap est composé des bundles Main, RabbitMQ Client, BasicProbes qui sont propres au système de déploiement. Le bundle Main est le point d'entrée du bootstrap. Le bundle RabbitMQ Client permet la liaison avec le domaine de déploiement, concrètement avec un serveur RabbitMQ [RabbitMQ, xx] qui gère le domaine (RabbitMQ est un middleware de communication par messages, cf. 2.2.5.2). Le bundle BasicProbes contient un ensemble de sondes basiques (cf. section suivante 2.2.5.1) ainsi présentes sur chaque appareil.

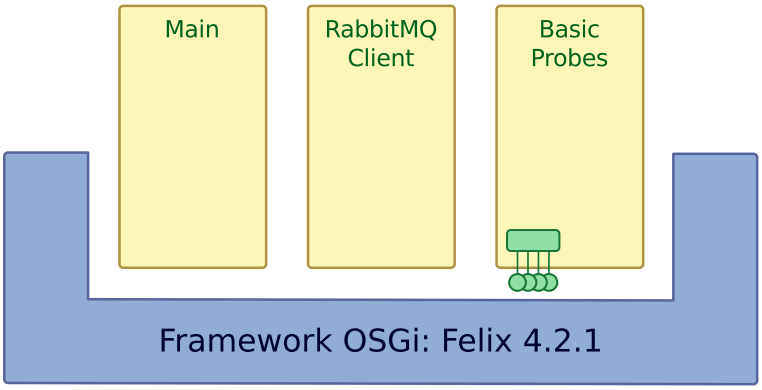

Figure 2.22 – Architecture du bootstrap

En installant le bootstrap, l'utilisateur qui est aussi l'administrateur du système donne les droits nécessaires au support local de déploiement. Le choix d'un framework OSGi (et par extension, de la plateforme Felix) permet de restreindre la phase de configuration du bootstrap demandée à l'utilisateur au seul choix entre deux programmes, en fonction du type d'appareil visé.

À titre d'illustration, les deux versions du bootstrap utilisables dans le projet INCOME sont disponibles à l'adresse suivante : `http://www.irit.fr/income/index.php?page=tache-5`. Pour les applications Android, le bootstrap est mis à disposition de l'utilisateur par l'intermédiaire d'un QR code, contenant l'adresse de téléchargement, lui évitant d'entrer manuellement cette adresse.

Sondes Le bootstrap contient un bundle de sondes basiques. Ces sondes sont celles qui sont les plus susceptibles d'être utilisées par le concepteur, permettant de récupérer des

informations logicielles et matérielles de l'appareil.

Les sondes basiques, implémentées avec l'aide d'étudiants de Master (projet de Master 1 et stage de Master 2), sont :

— CPU : mesure de la fréquence du processeur, etc.
— RAM : mesure de la mémoire vive libre, de mémoire vive totale, etc.
— HD : mesure de l'espace disque libre, espace disque total, etc.
— OS : reconnaissance du système d'exploitation, de la version, etc.
— Network : reconnaissance de l'adresse IP, du type de connexion, etc.
— Locale : identification des paramètres régionaux.
— Peripheral : liste des périphériques
— JVMMemory : mesure de la mémoire de la JVM (*Java Virtual Machine*, machine virtuelle Java) libre, totale, etc.
— Screen : liste des écrans, taille de l'écran, etc.
— Java : nom, version, vendeur, etc.

Le paquet fr.enac.lii.basicprobes est une implémentation du bundle OSGi décrit ci-dessus. Il permet d'offrir en REST [REST, xx] (sur l'hôte local, sur le port 8080 et à l'adresse */status/*, `localhost:8080/status`) l'ensemble des informations retournées par les sondes, dans un format JSON [JSON, xx].

Pour cela, il est nécessaire de disposer dans l'environnement OSGi des bundles qui apparaissent dans la figure 2.23. Jetty [Jetty, xx] est un conteneur de *servlets* (implémentation par défaut de conteneur de *servlets* dans Felix). L'enregistrement des *servlets* est simplifié en utilisant le service *whiteboard*, d'où le bundle correspondant. Pour une exécution correcte, Jetty a besoin d'un bundle gérant les logs : pour cela, nous avons choisi l'implémentation par défaut (Felix.log). La sérialisation des données au format JSON est réalisée par le framework Google Gson [Gson, xx] directement disponible sous forme de bundle OSGi.

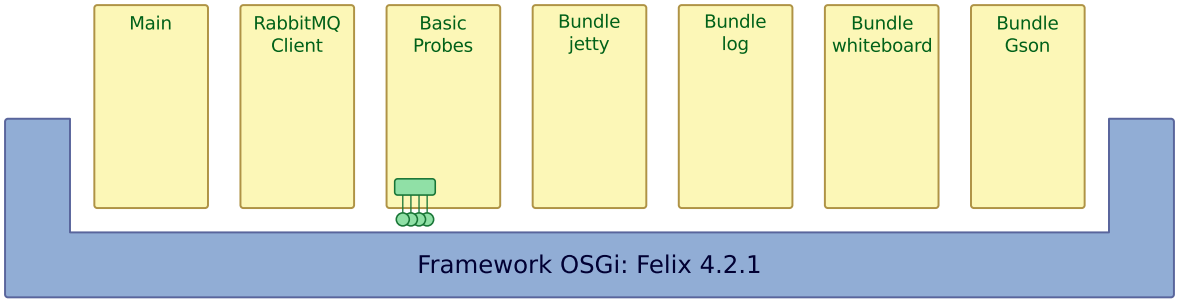

Figure 2.23 – Architecture complète du bootstrap

Android et J2SE Les deux versions du bootstrap sont une application native Android et une application Java standard (J2SE). Elles sont distribuées sous leurs formats natifs (.apk et .jar). Elles contiennent un moyen de démarrer l'implémentation d'OSGi choisie (Felix), et d'en contrôler l'exécution. C'est la seule partie de code du système de déploiement à être spécifique à une plateforme.

Dans les deux cas, le bootstrap utilise le système d'exploitation de l'appareil pour indiquer à l'utilisateur son état. On retrouve ainsi à l'exécution, dans la barre des tâches (ou l'équivalent sous Android), une icône permettant de vérifier l'état du système de déploiement, de l'arrêter, etc.

2.2.5.2 Inscription et enregistrement des appareils

Le gestionnaire de domaine de déploiement est dans notre cas un ensemble d'instances de serveurs RabbitMQ [RabbitMQ, xx] (protocole AMQP [AMQP, xx]) et un serveur agrégeant la liste des appareils disponibles. RabbitMQ permet de construire un protocole de présence et supporte la communication en mode asynchrone entre les appareils répartis à toutes les échelles du réseau (passage de firewalls/routeurs, etc.). Ainsi, il est possible de demander à distance l'installation, l'activation ou l'arrêt d'un bundle, ou encore l'état de l'appareil en invoquant le service de sondage basique.

Pour détecter la disparition, nous utilisons un système designalement *heartbeat* (battement de cœur). C'est un signal périodique qui permet de s'assurer de la présence des appareils dans le domaine de déploiement.

Nous avons réimplémenté un service de présence simple, dans lequel chaque client, identifié par un identifiant unique (UUID), dépose périodiquement cet identifiant dans la file de présence.

De ce fait, il faut également disposer d'un serveur (dont une instance tourne sur le serveur « flabelline.com » pour l'instant). Ultérieurement, il sera possible de faire tourner RabbitMQ sur des appareils sur le cloud. RabbitMQ est extensible par nature car il est possible de constituer un cluster de serveurs RabbitMQ. La librairie cliente assez légère (350 Ko) est livrée directement sous la forme d'un bundle OSGi.

La figure 2.24 présente les échanges entre le serveur RabbitMQ et le bundle Main du bootstrap.Lors de l'enregistrement, un thread de présence signale au serveur RabbitMQ la présence de l'appareil sur le domaine. Sur le serveur RabbitMQ, les évènements ont une durée de vie (TTL, *Time To Live*) de 60 secondes. Lorsque le moniteur de déploiement envoie une commande à un appareil, le serveur RabbitMQ envoie une commande *DMS Order* préfixée par l'UUID de l'appareil. Le bootstrap traite cette commande et renvoie une réponse, en préfixant toujours son message par son UUID, pour garder la traçabilité du traitement des commandes. Le *DMSShell* est un interpréteur de commandes qui permet le traitement des messages envoyés entre le bootstrap et le moniteur.

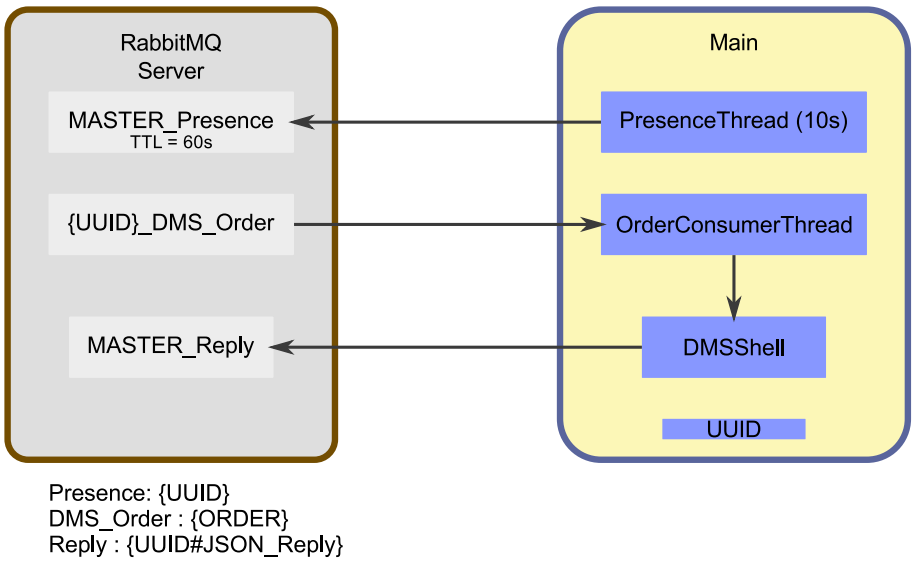

Figure 2.24 – Interactions entre le bundle Main et serveur RabbitMQ

2.2.5.3 Expression des propriétés de déploiement

Le DSL MuScADeL permet la spécification du déploiement des systèmes répartis multi-échelles. MuScADeL permet de décrire les types de composants à déployer, ainsi que leurs contraintes propres, les informations concernant les sondes utilisées pour la récupération de l'état du domaine, et les exigences de déploiement.

Un environnement de développement intégré de MuScADeL facilite l'expression du déploiement pour le concepteur. Un plugin Eclipse pour MuScADeL permet d'utiliser cet environnement. Le plugin a été conçu avec le framework Xtext [Xtext, xx]. Xtext permet, avec la spécification de la grammaire du langage d'obtenir un plugin avec coloration syntaxique, ainsi que des outils de validation du langage, comme la détection d'éléments non déclarés, l'auto-complétion sensible au contexte, etc. Le framework Xtext permet aussi la gestion de la séparation du code en plusieurs fichiers et l'inclusion de fichiers. Les fichiers MuScADeL ont pour extension *.musc*. Le plugin MuScADeL dans Eclipse est présenté dans la figure 2.25.

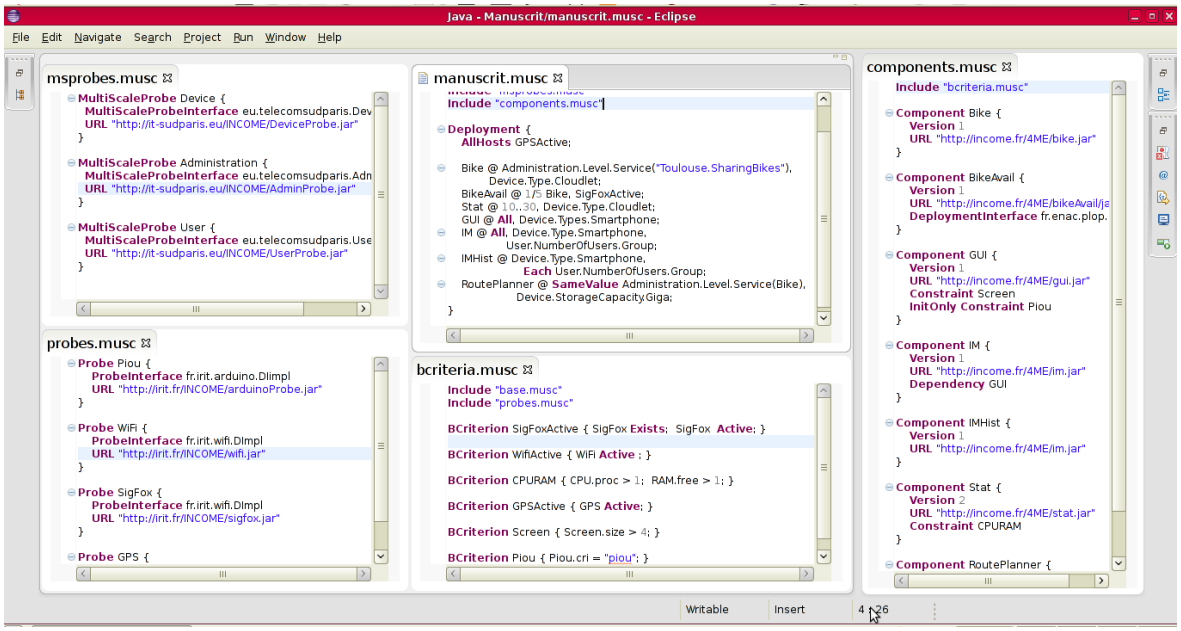

Figure 2.25 – Éditeur MuScADeL dans Eclipse

Le framework Xtext fournit aussi des outils pour personnaliser l'auto-complétion et la validation des éléments du langage. Nous l'utilisons pour les critères multi-échelles. Lorsque le concepteur ajoute dans le projet un fichier de modèles spécifique MuSCa (un fichier *.mscharacterization*), le plugin utilise ce fichier pour proposer une complétion des dimensions et des échelles, ainsi que pour valider leur utilisation (cohérence entre le point de vue, la dimension et l'échelle). La figure 2.26 montre cette complétion d'échelle.

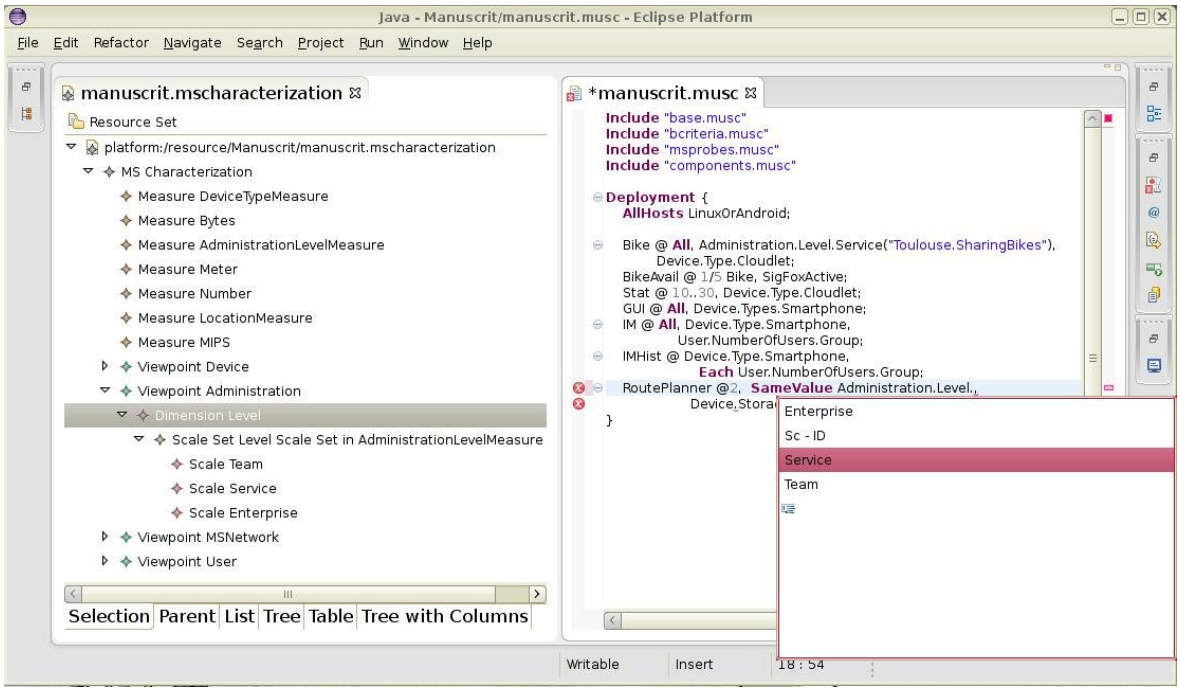

Figure 2.26 – Utilisation de la caractérisation MuSCa dans le plugin de MuScADeL

2.2.5.4 Analyse du descripteur MuScADeL

Le framework Xtend [Xtend, xx] permet la génération de code à partir de l'arbre syntaxique du langage défini. En parcourant l'arbre syntaxique de MuScADeL, nous générons du code Java qui sert à la récupération de l'état du domaine, à la construction des contraintes à donner en entrée au solveur de contraintes, et au lancement du déploiement. Pour cela nous utilisons une bibliothèque que nous avons définie, `MuScADeLDeploymentPlan`. Cette bibliothèque contient le code du DMSMaster, l'interface qui permet d'interagir avec le système de déploiement pour le déploiement initial. En fonction du descripteur MuScADeL, le code généré utilise cette bibliothèque pour préciser quelles sont les sondes utiles et quelles sont les propriétés définies afin d'ajouter les contraintes spécifiques. Tous les éléments du langage sont représentés dans cette bibliothèque.

Le code Java correspondant au descripteur MuScADeL est généré à chaque fois qu'un fichier MuScADeL est enregistré. Il est généré dans un projet existant appelé GenerateDeploymentPlan. Les noms des projets et fichiers MuScADeL sont réutilisés afin que le concepteur puisse retrouver le code correspondant à son projet courant.

2.2.5.5 Récupération de l'état du domaine

L'étape de récupération de l'état du domaine début par la récupération du descripteur des sondes utilisées et des critères et conditions à satisfaire. Ces deux éléments sont construits avec la génération du code Java, et donnés en entrée à la bibliothèque `MuScADeLDeploymentPlan`. Si des sondes spécifiques sont fournies par l'utilisateur, en plus des sondes basiques, la bibliothèque envoie une commande d'installation de ces sondes spécifiques au support local du déploiement sur les appareils. Une commande est aussi construite et envoyée afin de demander les informations de l'appareil. Cette commande

contient le nom de la sonde et de la méthode à appeler : nom de la méthode pour les sondes normales, dimension ou dimension/échelle pour les sondes multi-échelles. Chaque appareil envoie sa réponse. Puis, en fonction des valeurs contenues dans les conditions, une vérification de satisfaisabilité s'effectue sur le moniteur de déploiement. Cette étape de vérification permet la construction de la matrice *Dom*, qui est l'une des entrées de la phase de résolution de contrainte. Les commandes sont envoyées en utilisant les serveurs RabbitMQ.

2.2.5.6 Résolution des contraintes

La matrice *Dom* calculée à partir des résultats des sondages, et la matrice *Comp* à partir des données construites par le code généré, la phase de résolution peut être lancée. Pour cela, nous utilisons une bibliothèque que nous avons développée, `MuScADeLSolving`. Cette bibliothèque offre des méthodes pour l'ajout de propriétés de déploiement, qu'elle traduit dans des appels de méthodes. Ces appels de méthodes construisent le problème à résoudre pour un solveur de contraintes.

Solveur de contraintes Afin de pouvoir choisir un solveur de contraintes qui supporte la phase de résolution, nous avons étudié et comparé un certain nombre d'outils candidats : Cream [Tamura, 2003], Copris [Tamura, xx], JaCoP [Kuchcinski and Szymanek, xx], or-tools [ortools, xx], jOpt [jOpt, xx], et Choco [C.H.O.C.O. Team, 2010]. Nous avons considéré le type de problème traité et le support disponible en matière d'évolution et de documentation.

Notre problème étant un problème de satisfaction de contrainte (CSP), la classe de problème n'a pas été le critère de choix. Parmi ces solveurs, nous avons choisi **Choco** [C.H.O.C.O. Team, 2010]. Nous avons déjà une certaine expertise sur cet outil, et la librairie est complète et simple à utiliser.

`MuScADeLSolving` Dans cette section, nous présentons la librairie de résolution de contraintes issues des propriétés de déploiement `MuScADeLSolving`. L'interface `MuscadelSolvingInter` de la librairie (listing 2.6) présente les méthodes accessibles pour l'ajout des contraintes : `cardinaliteSimple`, `cardinaliteIntervalle`, `cardinaliteAll`, `ratio`, `sameValue`, `differentValue`, `each` et `solving`.

```
1  public interface MuscadelSolvingInter {
2      public void cardinaliteSimple (int cmp, int card);
3      public void cardinaliteAll (int cmp);
4      public void cardinaliteIntervalle(int cmp, int min, int max);
5      public void ratio(int cmpP, int cmpS, int ratioP, int ratioS);
6      public void sameValue(int cmp1, int cmp2, String[] sonde);
7      public void differentValue(int cmp1, int cmp2, String[] sonde);
8      public void each (int cmp, String[] sonde);
9      public int[][] solving() throws MuscadelSolvingExc;
10 }
```

Listing 2.6 – Interface `MuscadelSolvingInter`

La classe `MuscadelSolving` contient les matrices *Comp* et *Dom*, ainsi que le modèle Choco et la matrice de possibilités *SatVar*. Cette dernière est construite lors de la construction d'un objet `MuscadelSolving`, avec un appel à une méthode privée dans le constructeur.

La librairie est décrite plus en détails dans [Boujbel et al., 2014a].

Génération du plan de déploiement La construction des éléments du langage en utilisant la bibliothèque `MuScADeLDeploymentPlan` permet l'ajout transparent de propriétés de déploiement au problème Choco[22] en utilisant la bibliothèque `MuScADeLSolving`. En effet, chaque classe correspondant à une propriété contient une méthode qui ajoute la propriété représentée à l'instance de `MuscadelSolving`. Une simple boucle sur l'ensemble des instances correspondant aux propriétés permet l'ajout de l'ensemble des contraintes au problème du solveur de contraintes. Puis, la résolution est lancée. À la fin de cette étape, un plan de déploiement, sous la forme d'une matrice d'obligations $Composant * Appareil$ est disponible, et est présenté à l'utilisateur dans une fenêtre.

2.2.5.7 Installation et activation

Une fois le plan de déploiement obtenu, l'opérateur peut lancer le déploiement du système. En utilisant les serveurs RabbitMQ, une commande est envoyée aux différents appareils, contenant les différents composants à installer et à activer.

2.2.5.8 Déploiement dynamique

Le système de déploiement surveille le domaine et l'état du domaine à l'exécution de l'application. Si un nouvel appareil entre dans le domaine de déploiement, il s'enregistre sur le serveur RabbitMQ. Le moniteur de déploiement s'aperçoit qu'un nouvel appareil est dans le domaine de déploiement, il récupère ses caractéristiques en y installant les nouvelles sondes si nécessaires. Puis, il vérifie dans la liste des composants ayant une propriété dynamique si l'appareil satisfait toutes les propriétés du composant. Si oui, il déploie dessus le composant concerné. Pour le moment cette étape se fait de manière centralisée. À terme, cette vérification de satisfaction de propriétés se fera localement avec le support local du système de déploiement, et ce sera le superviseur hiérarchique qui piloterait l'installation et l'activation.

La surveillance de l'état du domaine permet de détecter les pannes provenant de la variation de l'état des appareils. Cette supervision peut se faire localement, en utilisant le framework de sondes. Le support local du système de déploiement réalise cette supervision, et disposant des valeurs seuils, il peut détecter la violation de propriété et en référer à son superviseur hiérarchique.

2.2.5.9 Conclusion

Nous avons présenté quelques éléments de l'implémentation de notre middleware MuScADeM. Il offre un environnement de développement intégré pour le langage dédié MuScADeL, sous la forme d'un plugin Eclipse. Le concepteur peut donc écrire le descripteur MuScADeL correspondant à son application et lancer directement la génération du plan de déploiement depuis Eclipse. Le support local de déploiement, utilisant un framework OSGi, permet aux appareils d'être visibles par le gestionnaire du domaine. Le gestionnaire de domaine tire parti de la scalabilité des instances RabbitMQ pour pouvoir traiter sans délai (sans engorgement) les nombreux appareils avec lequel il communique. Une fois le plan de

22. Le problème Choco est un ensemble de variables auxquelles des contraintes sont ajoutées. Dans notre cas, il s'agit des variables contenues dans la matrice *SatVar*.

déploiement généré, le concepteur initie, depuis Eclipse, le déploiement selon le plan initial. Le support local récupère donc le composant, l'installe et l'active. Une prise en compte des appareils entrant dans le domaine est aussi implémentée. Pour chaque nouvel appareil détecté par le gestionnaire de domaine, si cet appareil satisfait les propriétés d'un composant, ce composant est déployé dessus si besoin (c'est-à-dire si des propriétés dynamiques ont été définies).

Nous avons aussi effectué des tests de performance de la génération du plan de déploiement. Ces tests permettent de mettre en évidence certains écarts de temps de résolution entre les différentes propriétés exprimées. L'optimisation du traitement de ces propriétés demande un certain travail d'ingénierie.

2.2.6 Synthèse

Nous avons étudié le problème du déploiement des systèmes répartis multi-échelles. Nous en avons identifié les principales exigences en lien avec les propriétés caractéristiques d'hétérogénéité, de nombre, de dynamique et d'ouverture. Alors que le déploiement traditionnel est le plus souvent centralisé et dirigé par un opérateur humain, le déploiement des systèmes répartis multi-échelles doit être décentralisé, automatisé et sensible au contexte. Le rôle d'opérateur de déploiement doit y être joué par une application de niveau middleware : le système de déploiement.

Dans ce travail, nous avons proposé une solution qui s'articule autour de trois contributions principales : un processus de déploiement, un langage de spécification et un middleware pour la réalisation automatique.

Le processus de déploiement des systèmes répartis multi-échelles coordonne un certain nombre d'activités qui concernent la spécification du plan de déploiement, la génération et la réalisation du plan de déploiement initial, et la réalisation du déploiement dynamique. En particulier, il prend en compte l'ouverture du domaine de déploiement. La production du plan de déploiement est centralisée mais sa réalisation (déploiement initial et déploiement dynamique) est décentralisée et contrôlée dans le cadre d'une boucle « autonomique ». Pour cela, le processus s'appuie sur une infrastructure ouverte composée d'un support de spécification (un langage et son éditeur) et d'un middleware pour la mise en œuvre. Pour la sensibilité au contexte, le domaine et son état sont observés au moyen de sondes extraites de la spécification des propriétés du déploiement.

La conception du plan de déploiement est une activité particulière qui demande des moyens d'expression adéquats. Ces moyens sont donnés au concepteur sous la forme d'un langage dédié, MuScADeL (***Mu**lti**Sc**ale **A**utonomic **De**ployment **L**anguage*), qui offre un haut niveau d'abstraction et permet de s'abstraire des questions de mise en œuvre. Le concepteur du déploiement peut exprimer les contraintes d'exécution des composants ainsi que ses exigences en matière de distribution, de nombre et de dynamique du domaine. Pour spécifier le déploiement sans connaissance exacte du domaine et désigner des parties du domaine, il peut s'appuyer sur les échelles identifiées dans le système multi-échelle. MuScADeL est lié au métamodèle MuSCa de S. Rottenberg [Rottenberg et al., 2014], ce qui permet de contrôler la validité de l'expression des propriétés relatives aux échelles. Le concepteur peut également donner les éléments qui vont permettre l'acquisition d'un état personnalisé du domaine, en particulier il peut définir ses propres sondes. Ces sondes définies en fonction du besoin peuvent supporter des contrôles d'ordre divers : énergie, sécurité, qualité, etc. C'est cette possibilité de personnalisation qui permet de préserver la généricité du langage

et du middleware.

Pour l'automatisation du déploiement, nous avons proposé un middleware réparti, MuScADeM (***Mu**lti**Sc**ale **A**utonomic **De**ployment **M**iddleware*) qui supporte le déploiement initial de manière automatique, de la constitution du domaine et l'acquisition de son état jusqu'à la réalisation du plan de déploiement. L'architecture de MuScADeM comprend un gestionnaire de domaine qui enregistre les entrées et les sorties des appareils, un moniteur de déploiement et un support local de déploiement sur chaque appareil. Le support local est un conteneur de composants, initialement minimal, qui s'enrichit dynamiquement de composants pour la réalisation du déploiement. Il héberge à la fois les composants du système de déploiement et ceux de l'application déployée. Le moniteur de déploiement utilise un solveur de contraintes pour produire un plan de déploiement initial qui est à la fois adapté à l'état du domaine et conforme aux spécifications exprimées. Pour cela, les propriétés spécifiées sont transformées automatiquement en contraintes avant d'être traitées par le solveur.

Une fois le déploiement initial effectué, le système de déploiement doit maintenir en condition opérationnelle le système déployé en présence de perturbations. Pour réagir dynamiquement à ces perturbations, le système de déploiement réagit de manière autonomique. Nous avons identifié et analysé les différentes situations d'adaptation et, pour chacune, proposé une solution. Au vu des conditions d'exécution du système déployé (dynamique et taille du domaine), une architecture centralisée n'est pas envisageable. Nous proposons les éléments d'une architecture décentralisée qui permet une supervision par instance d'échelle et fournit, de manière autonomique, une réponse locale aux situations rencontrées.

Un prototype, qui satisfait l'essentiel des exigences énoncées, a été réalisé conformément à l'architecture définie. Un projet Master 1 ainsi qu'un stage de Master 2 y ont contribué en partie. Ce prototype prouve la faisabilité de nos propositions de processus et d'architecture. D'autre part, deux expériences ont montré que l'utilisation de nos outils est possible par des non spécialistes du déploiement dans le cadre de scénarios de déploiement particulièrement riches. Ces différents éléments participent ainsi à la validation expérimentale de nos solutions.

2.3 La vision multi-échelle

2.3.1 Problématique et motivations

La conception et le déploiement de systèmes répartis complexes sont des tâches difficiles qui demandent des outils pour les simplifier. En particulier, la forte hétérogénéité technique et la complexité de l'organisation des entités (sous-systèmes) peuvent amener les concepteurs et les développeurs à implémenter des solutions *ad hoc*, spécifiques à un système donné, qui ne peuvent pas être réutilisées pour la conception d'autres systèmes.

Nous nous sommes donc posé les questions suivantes : Comment faciliter la conception, le développement et le déploiement de systèmes répartis complexes ? En particulier, pouvons-nous proposer des méthodes et des outils qui permettent de concevoir des solutions adaptées non pas à un système donné mais à une famille de systèmes ? Notre proposition vise à exploiter et alimenter une base de connaissances et de technologies réutilisables pour de futurs systèmes.

Les principales caractéristiques des systèmes répartis complexes sont le grand nombre d'entités, la forte hétérogénéité technique, leur architecture dynamique, et les différents niveaux d'interactions entre entités. Nous remarquons que la complexité des systèmes répartis peut s'exprimer en termes d'échelles présentes dans ces systèmes, depuis différents points de vue (équipement, réseau, géographie, etc.). Les systèmes répartis complexes peuvent ainsi être vus comme des systèmes répartis multiéchelles. Cependant, des définitions, méthodes et outils, sont nécessaires pour mettre en œuvre cette nouvelle vision.

Par ailleurs, nous identifions qu'une approche de type descendante est appropriée pour maîtriser, voire réduire, la complexité de tels systèmes, et donc d'en faciliter la conception et le déploiement. Nous adoptons une des principales approches descendantes : l'Ingénierie Dirigée par les Modèles (IDM). Cette approche est intéressante car elle propose différents niveaux d'abstraction sous forme de modèles, et des transformations pour naviguer entre ces modèles. Cela permet aux concepteurs d'un système réparti complexe de maîtriser le niveau de complexité du système qu'ils conçoivent en sélectionnant un nombre restreint de concepts à prendre en compte dans la définition de chaque modèle. De plus l'IDM permet de rendre les modèles exploitables à l'exécution en proposant des mécanismes de génération de code à partir de modèles.

Dans ces travaux, nous proposons de répondre à cette problématique en nous appuyant sur la définition d'une vision multiéchelle des systèmes répartis complexes. Cette vision, accompagnée d'un vocabulaire spécifique, nous permet d'étudier un système réparti complexe selon différents points de vue et en considérant les échelles qui le composent. Au lieu de considérer les entités du système une par une, nous les regroupons en échelles pertinentes pour ce système, ce qui permet de réduire l'hétérogénéité présente dans le système : de milliers d'entités hétérogènes, nous passons à quelques dizaines d'échelles différentes. Nous proposons ensuite de rendre cette vision multiéchelle exploitable en la modélisant et en y appliquant la démarche IDM. Cette démarche met à disposition des concepteurs de systèmes répartis des outils utilisables à la fois au cours de la conception du système (modélisation du système à l'aide des concepts du multiéchelle), et à l'exécution du système (génération d'artefacts exécutables à partir d'un modèle du système).

Ainsi, nous proposons le *framework* MuSCa (***Mu**lti**S**cale **Ch**aracterization framework*) qui permet de caractériser la nature multiéchelle d'un système réparti complexe en identifiant

les différentes échelles présentes dans le système.

2.3.2 Processus de caractérisation multiéchelle

Dans ce chapitre, nous présentons le processus de caractérisation multiéchelle qui permet de proposer une vision simplifiée d'un système réparti complexe. Nous proposons un vocabulaire multiéchelle, nous envisageons l'étude d'un système complexe suivant plusieurs points de vue (par exemple, équipement et géographie), enfin nous énonçons le processus de caractérisation multiéchelle. Puis nous proposons une formalisation du vocabulaire multiéchelle et en particulier du concept d'échelle. Cette formalisation s'appuie sur des concepts mathématiques de la théorie des ensembles.

2.3.3 Concepts et processus de caractérisation multiéchelle

La définition d'un vocabulaire de caractérisation multiéchelle permet de partager un ensemble de concepts organisés. L'utilisation de ces concepts permet de construire une vision simplifiée des systèmes répartis multiéchelles.

Dans cette section, nous commençons par définir, dans la section 2.3.3.1, le vocabulaire multiéchelle proposé. Puis, dans la section 2.3.3.2, nous illustrons l'étude de la nature multiéchelle d'un système réparti complexe suivant différents points de vue identifiés. Nous continuons, dans la section 2.3.3.3, en soulignant l'existence de plusieurs types de dimensions. Enfin, nous énonçons le processus de caractérisation multiéchelle d'un système réparti complexe.

2.3.3.1 Vocabulaire multiéchelle

L'analyse multiéchelle d'un système réparti complexe proposée dans cette thèse repose sur les concepts de point de vue, dimension, mesure, échelle et ensemble d'échelles, introduits ci-dessous.

Point de vue En général, l'architecture d'un système est obtenue en étudiant ce système depuis différents *points de vue*. L'observation d'un système depuis un certain point de vue forme une *vue* du système [ISO/IEC/IEEE, 2011]. Une vue est constituée de *projections* des entités du système sur ce point de vue.

Dimension Chaque point de vue est étudié à travers différentes *dimensions*. Une dimension multiéchelle est une caractéristique mesurable d'une vue d'un système depuis un certain point de vue.

Mesure Une dimension est associée à une *mesure*, numérique ou sémantique. Dans le cas d'une mesure sémantique, les valeurs de cette mesure doivent être comparables entre elles et ordonnées.

Échelle À l'aide d'une mesure, une dimension peut être divisée en *échelles*. Une échelle correspond à un intervalle de valeurs de cette mesure. Les entités d'un système associées à une même échelle sont considérées comme homogènes pour le point de vue et la dimension considérés. Ainsi, afin de réduire la complexité présente dans un système, en terme d'hétérogénéité, les échelles doivent être suffisamment distinctes les unes des autres. Par exemple, pour une mesure numérique, les échelles peuvent

correspondre à des intervalles dont les centres sont distants de plusieurs ordres de grandeurs.

Ensemble d'échelles Un *ensemble d'échelles* contient les échelles choisies pour un couple dimension/mesure donné.

Remarque. Les termes *mesure*, *échelle*, et *ensemble d'échelles* ont été définis de manière plus formelle à l'aide de définition ensemblistes.

L'identification de plusieurs échelles pour un point de vue donné, dans une dimension donnée, avec une mesure donnée, permet de mettre en évidence la présence d'hétérogénéité dans le système réparti multiéchelle.

2.3.3.2 Étude de points de vue multiéchelles

Afin d'illustrer le processus de caractérisation multiéchelle, nous présentons dans cette section l'étude de plusieurs systèmes répartis multiéchelles depuis les points de vue suivants : équipement, donnée, réseau, géographie et utilisateur. Ces différents points de vue sont couramment identifiés dans l'étude des systèmes répartis complexes, comme par exemple dans les modèles de contexte [Dey et al., 2001a]. Pour chacun des points de vue, nous commençons par présenter un système qui met en avant le point de vue étudié. Pour ce système, nous sélectionnons une ou plusieurs dimensions, et pour chaque dimension nous proposons un ensemble d'échelles pertinentes pour la caractérisation multiéchelle du système étudié. Ainsi, nous illustrons que chaque système peut donner lieu à une caractérisation multiéchelle particulière.

1. Point de vue équipement

   En étudiant un système depuis le point de vue équipement, nous considérons les équipements qui le composent et faisons abstraction des autres composantes du système. Les auteurs de [Blair and Grace, 2012] présentent un scénario de surveillance de l'environnement, et plus particulièrement de surveillance de crue. Ce scénario fait intervenir une grande variété d'équipements. Des capteurs sont utilisés pour surveiller une rivière. Les données obtenues par les capteurs sont envoyées vers le *cloud* car le traitement de ces informations requiert une puissance de calcul très élevée. Enfin, les informations scientifiques et les alertes de sécurité sont accessibles depuis les équipements mobiles des différents acteurs du système (scientifiques, services d'urgence, tout public).

   Nous avons décidé d'étudier la nature multiéchelle de ce scénario, depuis le point de vue équipement, à travers deux dimensions, associées respectivement à deux mesures numériques. La première dimension est la capacité de stockage (en octets). La seconde est la puissance de calcul (en instructions par seconde). Dans la figure 2.27, nous positionnons certaines familles d'équipements par rapport à leur dispersion estimée pour ces deux mesures (capacité de stockage en haut et puissance de calcul en bas). Pour chaque dimension, nous choisissons plusieurs échelles pertinentes. Pour la dimension capacité de stockage, nous choisissions trois échelles : les échelles *kilo octets* en-dessous de $10^6 o$, *giga octets* entre $10^6 o$ et $10^{12} o$, et *peta octets* au-dessus de $10^{12} o$. Pour la dimension puissance de calcul, nous choisissons les échelles *kilo IPS* en-dessous de $10^6 IPS$, *giga IPS* entre $10^6 IPS$ et $10^{12} IPS$, et *peta IPS* au-dessus de $10^{12} IPS$. Chaque mesure est présentée avec une échelle logarithmique. Ainsi, nous

pouvons remarquer que les centres de deux échelles contiguës sont distants de plusieurs ordres de grandeurs.

Ces échelles représentent les ruptures de technologies existantes entre les équipements d'un système. Ainsi, deux équipements associés respectivement à deux échelles différentes, pour une dimension donnée, peuvent être considérés comme hétérogènes pour cette dimension, ce qui peut impliquer des fonctions différentes au sein du système. Par exemple, des équipements associés à des échelles différentes pour la dimension capacité de stockage peuvent être utilisés pour stocker ou manipuler des données de volumes significativement différents. De même, pour la dimension puissance de calcul, les différentes échelles peuvent être associées à différents types de calculs, qui requièrent plus ou moins de puissance.

À travers les deux dimensions étudiées, nous pouvons mettre en lumière la nature multiéchelle du système réparti de surveillance de crue depuis le point de vue équipement. En effet, pour ces deux dimensions, les équipements du système sont associés à plusieurs échelles. Par exemple, les capteurs, les équipements mobiles et les *clouds* sont associés respectivement aux échelles *kilo octets*, *giga octets* et *peta octets* pour la dimension capacité de stockage.

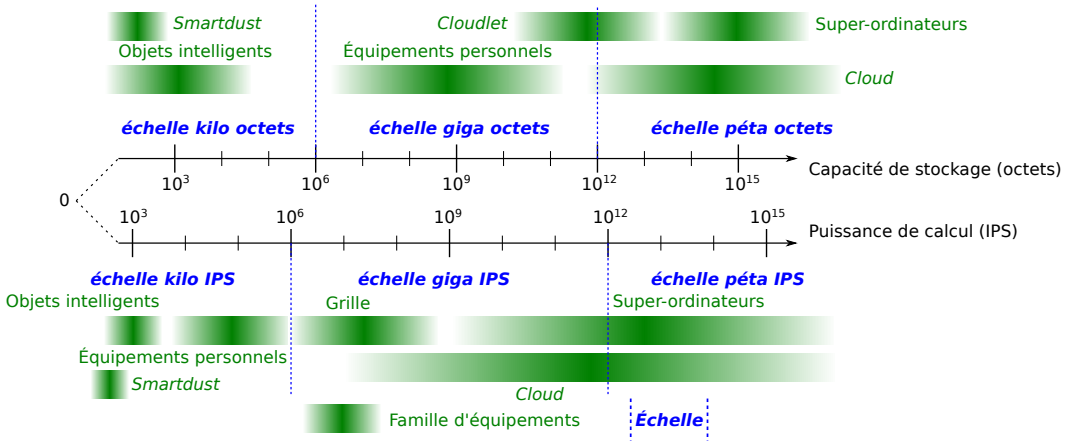

Figure 2.27 – Dimensions et échelles pour le point de vue équipement

2. Point de vue donnée

Le scénario de surveillance de crue [Blair and Grace, 2012] peut également être étudié suivant le point de vue donnée. Pour le point de vue donnée, nous considérons les données manipulées par le système. Les données provenant des capteurs sont transformées puis stockées dans le *cloud* afin de maintenir un historique des données et d'être disponibles pour des analyses statistiques. Dans le *cloud*, les données sont consolidées (c'est-à-dire analysées, structurées et synthétisées). Dans le scénario d'analyse de crues comme dans d'autres scénarios, dans le cadre de l'*IoT*, le volume des données présentes varie grandement depuis les données atomiques produites par des capteurs individuels jusqu'aux données *big data* obtenues par le rassemblement d'un grand nombre de capteurs, potentiellement répartis dans le monde entier, et stockées dans le *cloud*.

Comme le montre la figure 2.28, pour le point de vue donnée, nous avons choisi la dimension volume de données, mesurée en octets. Pour cette dimension nous identifions deux échelles pertinentes pour le scénario de surveillance de crue : l'échelle *très petit volume*, en-dessous de $10^4 o$, pour les données de capteur, et l'échelle *très gros*

volume, au-dessus de $10^{14}o$, pour l'historique des données. Entre ces deux échelles extrêmes, explicites dans le scénario de surveillance de crue, nous pouvons identifier deux autres échelles qui nous semblent pertinentes pour d'autres systèmes : l'échelle *petit volume*, entre 10^4o et 10^7o, et l'échelle *gros volume*, entre 10^7o et $10^{14}o$.

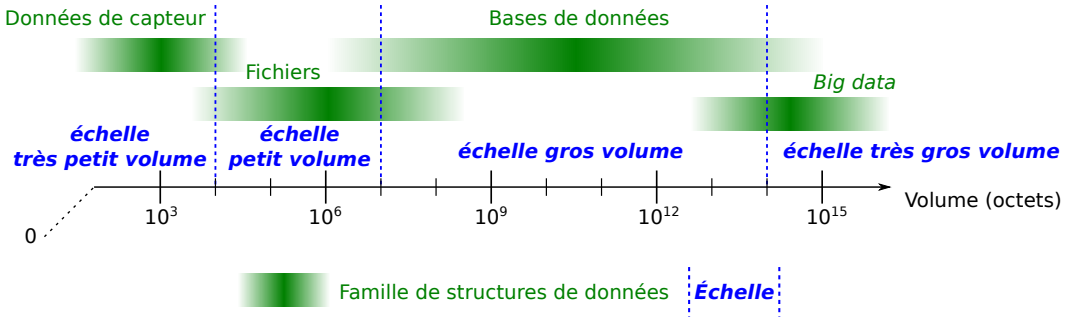

Figure 2.28 – Dimensions et échelles pour le point de vue donnée

3. Point de vue réseau

   Pour étudier un système depuis le point de vue réseau, nous considérons les réseaux traversés par le système. Les auteurs de [Kessis et al., 2009] proposent un middleware pour la gestion de ressources en contexte multiéchelle. Le système proposé a pour but de permettre une gestion flexible de ressources. Le terme multiéchelle est utilisé pour décrire la répartition des ressources sur différents réseaux de différentes portées : des réseaux personnels (*Personal Area Network*, PAN), des réseaux locaux (*Local Area Network*, LAN), et des réseaux étendus (*Wide Area Network*, WAN).

   Nous illustrons la nature multiéchelle du point de vue réseau à travers trois dimensions, associées respectivement à trois mesures numériques. La figure 2.29 illustre ces trois dimensions : la dimension latence mesurée en millisecondes, la dimension bande passante mesurée en kilobits par seconde et la dimension portée mesurée en mètres. Puis nous positionnons certaines familles de réseaux (par exemple, zigbee, WiFi, satellite) par rapport à la répartition estimée de ces familles pour les trois mesures considérées. Ensuite, pour chaque couple dimension/mesure, nous identifions des échelles pertinentes. Pour la latence, nous identifions deux échelles : l'échelle de *faible latence* en-dessous de $1s$ et l'échelle de *forte latence* au-dessus de $1s$. Pour la bande passante, nous identifions trois échelles : l'échelle de *faible bande passante* en-dessous de $10^3kb/s$, l'échelle de *moyenne bande passante* entre 10^3 et $10^6kb/s$, et l'échelle de *forte bande passante* au-dessus de $10^6kb/s$. Enfin, pour la portée, nous identifions les échelles PAN, LAN, MAN (Metropolitan Area Network), et WAN.

4. Point de vue géographie

   Le point de vue géographie est pertinent pour étudier la nature multiéchelle d'un système réparti complexe. Pour ce point de vue, nous considérons les coordonnées GPS des entités du système. Les auteurs de [Franklin et al., 2005] présentent une base de données hautement répartie pour l'*IoT*. Dans ce système, le point de vue géographie est utilisé pour définir différentes zones de différentes échelles afin de construire un système hiérarchique. Pour chaque zone, un ordinateur dédié est responsable du traitement des requêtes et de la gestion des données : par exemple, pour les zones ambiantes, régionales, ou pour le monde entier.

   Nous illustrons le point de vue géographie à travers deux dimensions, comme le montre la figure 2.30, l'une associée à une mesure numérique et l'autre à une me-

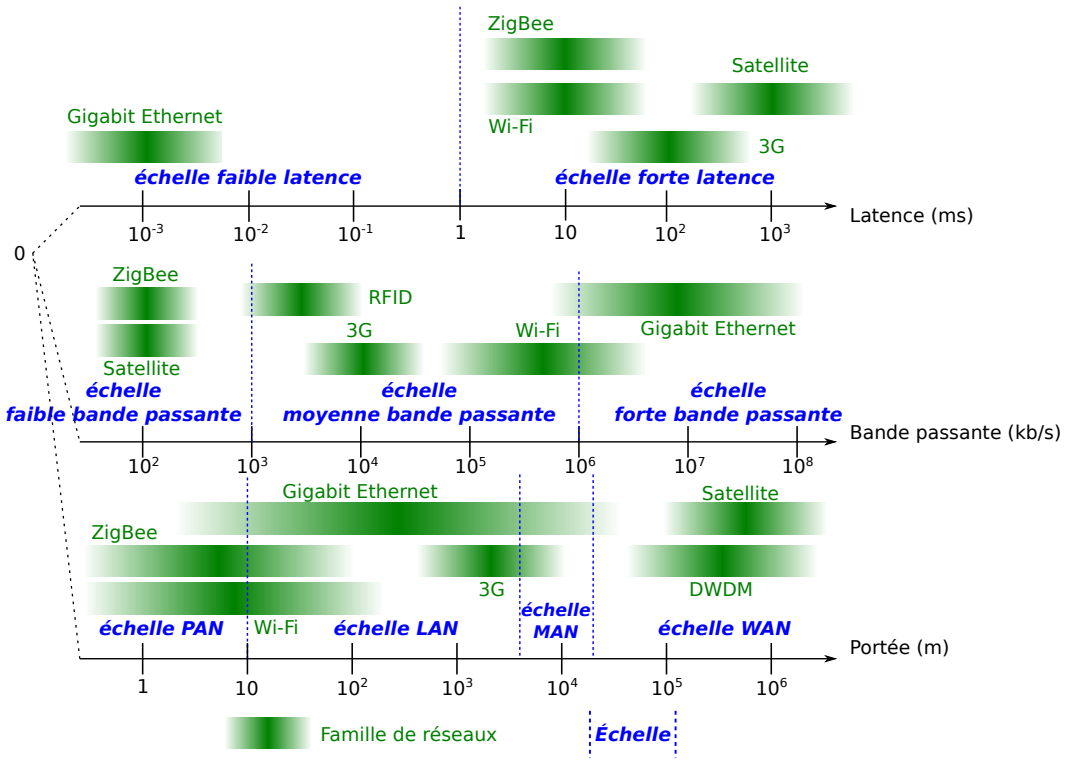

Figure 2.29 – Dimensions et échelles pour le point de vue réseau

sure sémantique. La première dimension est la dimension distance qui mesure en mètres la distance maximale entre deux éléments d'un ensemble d'entités. Pour cette mesure numérique, nous avons choisi un ensemble de quatre échelles : l'échelle *locale* en-dessous de $10m$, l'échelle *accessible à pied* entre $10m$ et 10^3m, l'échelle *accessible en voiture* entre 10^3m et 10^5m, et l'échelle *accessible en avion* au-dessus de 10^5m. La deuxième dimension est la dimension localisation administrative, qui est associée à une mesure sémantique. Elle s'applique également à un ensemble d'entités et mesure la plus petite localisation commune à cet ensemble d'entités. Nous avons retenu un ensemble de six échelles : l'*immeuble*, le *quartier*, la *ville*, la *région*, le *pays*, et le *monde*. L'organisation d'un système et les interactions entre les entités de ce système peuvent varier en fonction de la dimension et des échelles choisies dans le point de vue géographie. D'autre part, comme le suggère le scénario présenté section `#sec:scenario`, les échelles définies pour la dimension localisation administrative peuvent être utilisées pour spécifier qu'un certain composant doit être installé dans chaque ville, ou dans chaque quartier.

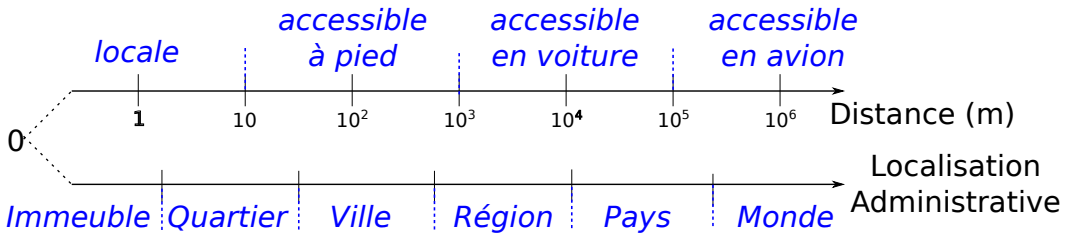

Figure 2.30 – Dimensions et échelles pour le point de vue géographie

5. Point de vue utilisateur

   Le dernier point de vue présenté est le point de vue utilisateur. Pour ce point de vue, nous considérons des ensembles d'utilisateurs qui accèdent au système. Pour illustrer ce point de vue, nous prenons pour exemple le système de gestion de contexte proposé par le projet ANR INCOME [Arcangeli et al., 2012]. Dans ce système, pour des raisons de protection de la vie privée, il est nécessaire de prendre en compte différentes catégories d'utilisateurs et d'ensembles d'utilisateurs. En effet, une information a des propriétés de confidentialité différentes en fonction des catégories d'utilisateurs auxquels elle est présentée. Le système considère différemment un utilisateur isolé, les groupes, les communautés, et le reste du monde.

   Comme le montre la figure 2.31, pour ce système, nous avons choisi d'étudier le point de vue utilisateur à travers deux dimensions. La première dimension est mesurée par le nombre d'utilisateurs qui composent un ensemble d'utilisateurs du système. Pour cette dimension nous avons identifié quatre échelles : l'*individu* pour 1 utilisateur, le *groupe* entre 2 et 10^2, la *communauté* entre 10^2 et 10^5, et *tous* au-dessus de 10^5. La deuxième dimension est associée à une mesure sémantique : nous considérons la politique d'accès utilisée quand un nouvel utilisateur souhaite joindre l'ensemble d'utilisateurs. Nous considérons plusieurs politiques d'accès : impossible (l'ensemble d'utilisateurs est fermé), contrôlé (il est possible de joindre l'ensemble avec une autorisation), ouvert (l'ensemble peut être rejoint part une simple inscription), et public (aucune inscription n'est nécessaire). Chaque politique d'accès est associée à une échelle : *individu*, *groupe*, *communauté*, et *tous*.

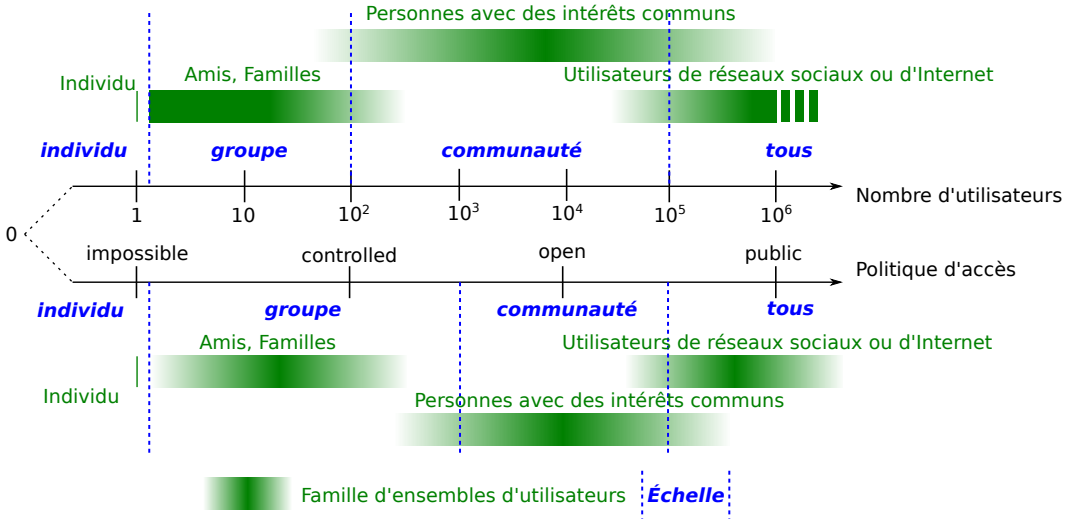

Figure 2.31 – Dimensions et échelles pour le point de vue utilisateur

2.3.3.3 Typologie des dimensions multiéchelles

À partir des exemples précédents, nous distinguons les dimensions multiéchelles hiérarchiques et les non hiérarchiques.

Par exemple, si nous considérons la dimension localisation administrative, pour le point de vue géographie, et la dimension capacité de stockage, pour le point de vue équipement, nous remarquons que les échelles pertinentes construites pour ces dimensions ne sont pas

organisées de la même manière. En effet, pour la dimension localisation administrative (cf. figure 2.30), les échelles sont organisées les unes *dans* les autres, c'est-à-dire de manière hiérarchique : un quartier *est dans* une ville, une ville *est dans* une région, etc. À l'inverse, les échelles de la dimension capacité de stockage (cf. figure 2.27) sont organisées les unes *à côté* des autres, c'est-à-dire de manière non hiérarchique : on ne peut pas dire qu'un équipement de l'échelle kilo octets *est dans* un équipement de l'échelle giga octets.

Cette distinction entre dimensions hiérarchiques et non hiérarchiques nous amène à envisager deux façons de caractériser les entités d'un système de façon multiéchelle. La méthode la plus intuitive consiste à mesurer chaque entité du système de façon indépendante. Par exemple, pour le point de vue équipement et pour l'ensemble d'échelles choisi dans la figure 2.27, chaque équipement du système peut être associé à une échelle en fonction de sa capacité de stockage. Notons qu'il est possible qu'une entité soit associée à plusieurs échelles si les échelles de l'ensemble d'échelles se chevauchent. Cependant, dans le cas d'une dimension hiérarchique, les échelles étant organisées les unes *dans* les autres, une entité du système peut être associée à toutes les échelles d'un ensemble d'échelles. Par exemple, pour la dimension localisation administrative du point de vue géographie, et pour l'ensemble d'échelles choisi dans la figure 2.30, toute entité du système qui peut être associée à un immeuble est également associée à un quartier, une ville, une région, etc. C'est pourquoi, dans le cas d'une dimension hiérarchique, nous proposons des mesures appliquées à un ensemble d'entités. Par exemple, si l'on considère trois entités d'un système, l'une située à Toulouse, et les deux autres à Paris, nous pouvons associer l'ensemble de ces trois entités à l'échelle pays, qui est la plus petite échelle commune à ces trois entités.

2.3.4 Énoncé du processus de caractérisation multiéchelle

L'étude précédente nous permet d'énoncer le processus de caractérisation multiéchelle.

Processus de caractérisation multiéchelle. La nature multiéchelle d'un système réparti complexe doit être étudiée indépendamment depuis chacun des points de vue choisis par le concepteur du système. Pour chaque point de vue, une vue restreinte du système est considérée. Puis, pour cette vue, une ou plusieurs dimensions sont choisies. Afin d'identifier les échelles des dimensions étudiées, chaque dimension est associée à une mesure numérique ou sémantique. Les échelles sont des intervalles de valeurs de cette mesure. Le choix des points de vue, des dimensions/mesures, et des échelles pertinentes pour une caractérisation multiéchelle reste ouvert. C'est au concepteur du système de choisir en fonction des propriétés du système qu'il veut étudier. Ces choix lui permettent de mettre en évidence des caractéristiques spécifiques à gérer au cours des phases de conception et/ou d'exécution du système.

Nature multiéchelle d'un système. La nature multiéchelle d'un système est relative à une caractérisation multiéchelle. Pour un point de vue donné, et pour un couple dimension/-mesure, chaque projection d'entité —ou ensemble de projections d'entités, dans le cas d'une dimension hiérarchique— d'une vue restreinte du système est associée à une échelle. Pour une caractérisation donnée, et un point de vue donné, un système distribué est qualifié de multiéchelle quand, pour au moins une dimension, les projections d'entités —ou ensembles de projections d'entités— de sa vue sont associées à des échelles différentes. L'identification de la nature multiéchelle d'un système met en évidence la présence d'hétérogénéité (technique ou non technique) pour ce point de vue, cette dimension et cette mesure.

2.3.4.1 Conclusion

Dans cette partie, nous avons tout d'abord présenté les concepts du vocabulaire multiéchelle que nous proposons. Nous avons défini un processus de caractérisation multiéchelle pour les systèmes répartis complexes. Par ailleurs, nous avons égalment proposé une formalisation des concepts d'échelle et d'ensemble d'échelles à l'aide de la théorie des ensembles, ce travail est discuté en détail dans la thèse de Sam Rottenberg. Cette formalisation nous permet notamment de proposer des contraintes que doit respecter un ensemble d'échelles afin de posséder des propriétés qui facilitent la caractérisation multiéchelle d'un système. En effet, ces propriétés permettent, par exemple, d'assurer que, pour chaque dimension de chaque point de vue de la caractérisation, toute entité du système pourra être associée à au moins une échelle, voire à une unique échelle dans le cas où l'ensemble d'échelles forme une partition de la mesure considérée. Par ailleurs, la propriété d'ordre entre les échelles d'un ensemble d'échelles permet de pouvoir comparer les échelles entre elles, ce qui augmente l'expressivité du vocabulaire multiéchelle en permettant, par exemple, d'exprimer des contraintes qui concernent l'ensemble des entités associées à toutes les échelles *supérieures à* ou *inférieures à* telle échelle.

Les résultats de ce chapitre constituent les fondements du *framework* MuSCa que nous présentons ci-après.

2.3.5 Réalisation

Ce *framework* a pour but, d'une part, de mettre en œuvre le processus de caractérisation multiéchelle, et d'autre part de produire des artefacts logiciels adaptés à une caractérisation.

Le *framework* MuSCa propose, d'une part, une méthodologie pour la phase de conception des systèmes répartis complexes basée sur l'analyse multiéchelle de l'architecture de ces systèmes, et d'autre part, un mécanisme de production d'artefacts logiciels qui apportent la conscience des échelles aux composants de ces systèmes au cours de la phase d'exécution. Nous avons choisi de proposer un *framework* qui s'appuie sur l'IDM, car cette démarche, et l'ensemble des outils existants qui l'implémentent (générateurs de code, langages de transformation de modèles, etc.) permettent d'établir un lien direct entre l'application du processus de caractérisation multiéchelle, au cours de la conception d'un système, et la production d'artefacts exécutables.

La phase de conception, détaillée sous forme d'un diagramme SPEM [OMG, 2008b], est composée de deux activités principales.

La première activité est la caractérisation multiéchelle, qui consiste à abstraire une vision multiéchelle d'un futur système distribué complexe. Afin de guider le concepteur du système et de capitaliser sur les anciennes caractérisations, elle prend en entrée un document partagé que nous appelons la taxonomie multiéchelle. Cette taxonomie contient l'ensemble des éléments de caractérisation ayant déjà été utilisés pour d'anciennes caractérisations. De plus elle est extensible, ce qui permet au concepteur du système d'ajouter les éventuels nouveaux termes de caractérisation dont il peut avoir besoin, contribuant ainsi à la capitalisation d'une connaissance partagée. Ces termes seront alors accessibles pour de futures caractérisations. Cette première activité produit en sortie une caractérisation multiéchelle du système étudié, sous forme d'un modèle, qui correspond à une restriction de la taxonomie multiéchelle.

La deuxième activité est la génération d'artefacts logiciels qui apportent la conscience multiéchelle aux composants du système à l'exécution. Nous appelons ces artefacts des sondes multiéchelles. Cette activité prend en entrée le modèle de caractérisation multiéchelle produit par la première activité, ainsi que des sondes logicielles de bas niveau. Ces dernières sont des programmes qui s'exécutent sur les entités du système afin de mesurer les caractéristiques de ces entités dans différentes dimensions. Par exemple, pour le point de vue équipement, une sonde de bas niveau peut retourner la capacité de stockage ou la puissance de calcul d'un équipement. Nous considérons que, pour chaque dimension choisie dans la caractérisation d'un système, il existe une sonde de bas niveau qui mesure cette dimension. Les sondes multiéchelles générées au cours de cette deuxième activité consolident les données fournies par les sondes de bas niveau afin d'apporter la conscience des échelles aux composants du système à l'exécution.

2.3.5.1 Éditeur de caractérisation multiéchelle

Nous avons implémenté le *framework* MuSCa à l'aide du *Eclipse Modeling Framework Project* [23] (EMF). Les métamodèles MuSCa et MuSCa_Probes sont définis comme des instances du métamétamodèle Ecore. EMF génère automatiquement un éditeur de modèles MuSCa. Nous avons étendu cet éditeur pour y ajouter un assistant qui permet de créer un modèle de caractérisation en sélectionnant des points de vue, dimensions, et ensembles d'échelles présents dans la taxonomie MuSCa (cf. figures 2.32, 2.33, et 2.34). Ce modèle peut ensuite être complété via l'éditeur généré par EMF.

L'éditeur de caractérisation multiéchelle est empaqueté sous la forme d'un *plugin* [24] de l'environnement de développement Eclipse [25].

23. http://www.eclipse.org/modeling/emf/
24. Nous utilisons le terme anglais « plugin » qui est plus couramment utilisé que l'expression française « module d'extension ».
25. https://eclipse.org/

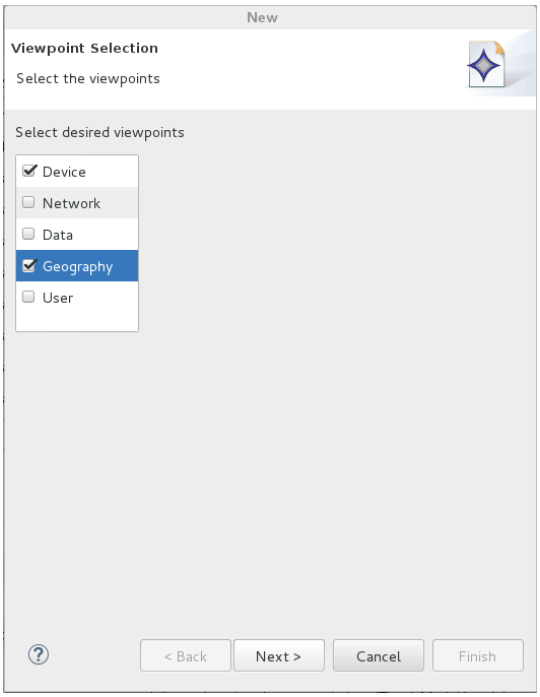

Figure 2.32 – Assistant de sélection des points de vue

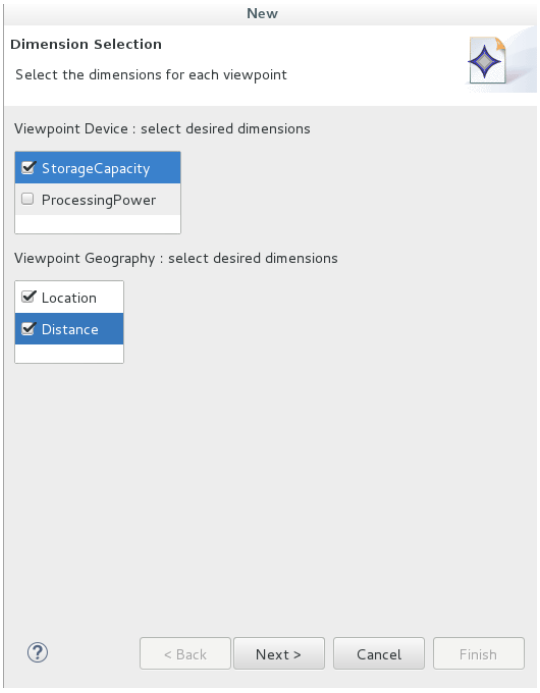

Figure 2.33 – Assistant de sélection des dimensions

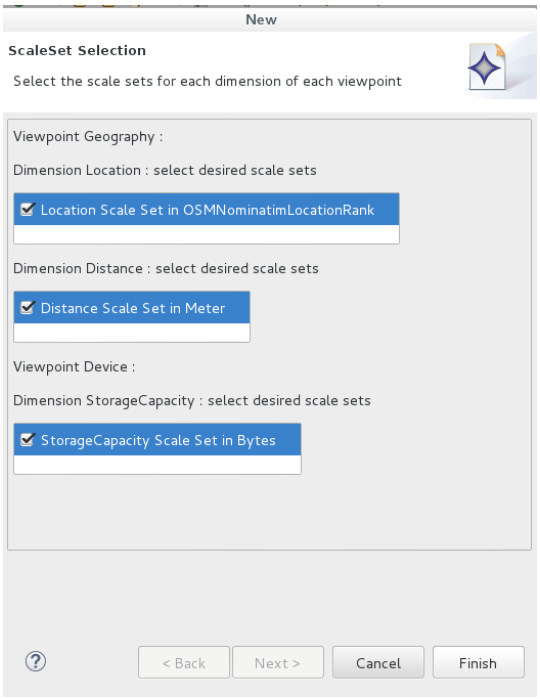

Figure 2.34 – Assistant de sélection des ensembles d'échelles

2.3.5.2 Conscience des échelles

Pour générer les sondes multiéchelles nous utilisons le générateur de code Acceleo[26] qui implémente le standard MOF Model To Text Transformation Language (MOFM2T) de l'OMG [OMG, 2008a]. Pour cela nous avons défini des *templates* Acceleo qui contiennent une implémentation générique des sondes multiéchelles. Ces *templates* sont complétés par le générateur de code pour produire le code des sondes multiéchelles correspondant à un modèle donné en entrée. Nous avons choisi de générer des sondes implémentées en Java. Les *templates* ainsi que les classes nécessaires au lancement de la génération de code sont empaquetés sous la forme d'un deuxième *plugin* pour l'environnement Eclipse.

Après avoir installé ce *plugin*, la génération de code peut être lancée en accédant au menu contextuel (clique droit) lié à un modèle de caractérisation multiéchelle (cf. figure 2.35).

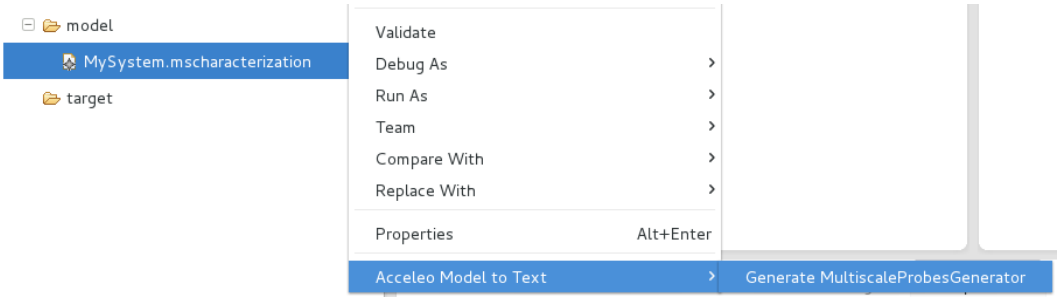

Figure 2.35 – Menu contextuel Acceleo

Un extrait d'une sonde automatiquement générée est présenté dans la figure 2.36. Cette figure présente l'énumération générée pour l'ensemble d'échelles associé à la dimension distance et à la mesure mètres. Cette énumération contient notamment, lignes 2 à 5, les différentes échelles qui peuvent être retournées par la sonde multiéchelle pour l'ensemble d'échelles correspondant. Elle contient également deux méthodes : une méthode statique `getScale`, ligne 15, qui retourne la première échelle contenant la valeur mesurée passée en paramètre, et une méthode `isInScale`, ligne 24, qui détermine si une valeur mesurée est incluse dans une échelle.

Remarque. La méthode *`getScale`* retourne toujours la première échelle trouvée. Cependant, il est possible de la modifier après génération afin d'implémenter une méthode de levée d'ambiguïté lorsque plusieurs échelles contiennent une même valeur mesurée.

La figure 2.37 présente un extrait de *template* Acceleo.

Évaluation du nombre de lignes générées Pour une caractérisation contenant trois ensembles d'échelles (associés aux dimensions distance et localisaton administrative pour le point de vue géographie, et à la dimension capacité de stockage pour le point de vue équipement), le processus de génération de sondes multiéchelles produit 991 lignes de code.

26. http://www.eclipse.org/acceleo/

```
1  public enum Distance_In_Meter_Scale {
2      LOCAL("0", "10"),
3      FOOTDISTANCE("0", "1000"),
4      CARDISTANCE("0", "100000"),
5      PLANEDISTANCE("0", "Infinity");

7      private final Meter_Value min;
8      private final Meter_Value max;

10     Distance_In_Meter_Scale(String min, String max) {
11   this.min = new Meter_Value(min);
12   this.max = new Meter_Value(max);
13     }

15     public static Distance_In_Meter_Scale getScale(Meter_Value value) {
16   for (Distance_In_Meter_Scale scale : Distance_In_Meter_Scale.values()) {
17       if (scale.isInScale(value)) {
18     return scale;
19       }
20   }
21   return null;
22     }

24     private boolean isInScale(Meter_Value value) {
25   return (value.compareTo(this.min) >= 0) && (value.compareTo(this.max) <= 0);
26     }
27 }
```

Figure 2.36 – Extrait de sonde générée : énumération Distance_In_Meter_Scale

```
1 [for (scale : Scale | aScaleSet.scales) separator(',\n')]
2     [scale.name.toUpper()/]("[scale.min/]", "[scale.max/]")
3 [/for];
```

Figure 2.37 – Extrait de *template* Acceleo

2.3.6 Synthèse

Dans cette partie, nous avons présenté processus de caractérisation multiéchelle ainsi que le *framework* MuSCa. Ce *framework* repose sur une démarche d'IDM.

Premièrement, il permet au concepteur d'un système réparti complexe de mettre en œuvre le processus de caractérisation multiéchelle à l'aide d'un éditeur de caractérisation multiéchelle. Cet éditeur, qui fait partie du *framework* MuSCa, a été généré à partir d'un métamodèle qui constitue une représentation formelle et informatisée des concepts du vocabulaire multiéchelle que nous avons proposé dans le chapitre précédent. Pour guider le concepteur, l'éditeur propose un assistant qui intègre une taxonomie multiéchelle qui contient tous les éléments de caractérisation déjà identifés lors de précédentes caractérisations. Le concepteur a également la possibilité d'étendre cette taxonomie. Grâce à cet éditeur, un concepteur de système peut définir une caractérisation multiéchelle du système qu'il souhaite concevoir. Une telle caractérisation est un modèle conforme au métamodèle MuSCa.

Deuxièmement, à partir d'un modèle de caractérisation MuSCa, le *framework* permet de

générer des artefacts logiciels, que nous appelons des sondes multiéchelles, qui apportent la conscience des échelles aux composants du système caractérisé. Le *framework* met donc en œuvre la démarche IDM en rendant productif un modèle de caractérisation. Les sondes multiéchelles sont générées spécifiquement pour une caractérisation donnée, et donc pour un système donné, ce qui permet de garantir la concordance des échelles pouvant être retournées par les sondes avec les échelles définies dans une caractérisation. Ces sondes multiéchelles sont déployées à travers l'ensemble du système et permettent aux entités, ou ensembles d'entités, de savoir à tout moment à quelle(s) échelle(s) elles ou ils sont associé(e)s.

Le respect d'une démarche IDM permet d'augmenter l'intérêt du processus de caractérisation multiéchelle en rendant productifs les modèles de caractérisation. De plus, l'IDM garantit une évolutivité aisée du *framework*. Par exemple, s'il est nécessaire de modifier le métamodèle MuSCa, l'éditeur de caractérisation peut être automatiquement regénéré et mis à jour pour refléter ces changements. Par ailleurs, de nombreux outils existent pour implémenter cette démarche, ce qui réduit l'effort de développement nécessaire à l'implémentation du *framework*.

Nous avons validé qualitativement notre démarche à travers trois cas d'utilisation réalisés dans le cadre du projet ANR INCOME. Le premier cas d'utilisation concernait la caractérisation multiéchelle de différents scénarios de l'Internet des objets. Il a permis de valider d'une part l'utilisation de la caractérisation multiéchelle comme une grille de lecture pour ces scénarios, et d'autre part l'extensibilité de la taxonomie MuSCa. Le deuxième cas d'utilisation portait sur le déploiement multiéchelle, et le troisième concernait le filtrage et le routage d'informations de contexte au sein d'un gestionnaire de contexte multiéchelle. Ces deux cas d'utilisation ont permis de valider d'une part l'expressivité du vocabulaire multiéchelle, à travers la définition de contraintes, et d'autre part l'utilisation des sondes multiéchelles générées pour l'application de ces contraintes. Ces travaux de validation ont montré la concordance des contributions de cette thèse avec les exigences du projet INCOME. Les travaux réalisés dans le cadre de la thèse, et en particulier l'implémentation du *framework* MuSCa, sont aujourd'hui entièrement intégrés au projet INCOME.

3 Projet de recherche

3.1 Motivation pour l'HDR

L'Habilitation à Diriger des Recherches, est pour moi un objectif double. C'est évidemment le sésame qui permet l'encadrement doctoral, probablement une des activités de recherche qui me motive le plus. A mon goût, travailler avec un doctorant permet de tenir une trajectoire relativement stable sur le plan de la recherche et surtout de prendre du temps de réflexion, ce qui apparaît parfois comme un luxe dans l'agenda chargé par la schizophrénie des fonctions de l'enseignant-chercheur.

L'HDR aussi un moyen de renforcer la visibilité et la dynamique de la recherche au sein de L'École Nationale de l'Aviation Civile. Cette grande école d'ingénieur française n'a pas la même histoire du point de vue de l'organisation et de la visibilité de la recherche que les établissements traditionels. A l'origine, elle compte dans son personnel permanent des ingénieurs formés au sein de l'établissement et dont les activités de recherche sont dédiée à la construction des systèmes du contrôle aérien [1]. Peu d'entre eux disposent d'une thèse de doctorat (encore moins d'une habilitation à diriger les recherches !) et dans l'esprit de compétitivité actuel, l'ENAC accorde une grande importance au recrutement externe d'enseignants chercheurs docteurs, et par suite au passage de l'HDR.

Dans mon équipe d'accueil, seule une personne est titulaire de l'HDR aujourd'hui. Cela nous limite dans les réponse aux appels d'offre ou aux bourses permettant de financer des doctorants, et lorsque nous arrivons à obtenir certains de ces financements, nous devons faire appel fréquemment à des encadrants/directeurs de thèse extérieurs. Ce qui ne permet pas d'avoir une qualité d'encadrement idéale, ni un encadrement parfaitement spécialiste des contextes dans lesquels les recherches se déroulent. En espérant que cela puisse changer !

3.2 Thématique principale de l'équipe ENAC/LII

L'École Nationale de l'Aviation Civile (ENAC), en tant qu'établissement d'enseignement supérieur sous tutelle de la DGAC, oriente les travaux de recherche de ses enseignants-chercheurs sur les sujets dont elle anticipe l'importance pour l'industrie aéronautique et pour la sécurité aérienne.

Le Laboratoire d'Informatique Interactive (LII) est spécialisé dans l'interaction homme-machine et dans ses aspects informatiques. Le laboratoire s'intéresse en particulier d'une

1. https://fr.wikipedia.org/wiki/Centre_d%27%C3%A9tudes_de_la_navigation_a%C3%A9rienne

part à l'ingénierie des systèmes interactifs, rencontre de l'ingénierie des systèmes et de la conception des logiciels interactifs ; d'autre part à la modélisation des logiciels interactifs, à la fois pour proposer des langages et architectures logicielles adaptés, et pour progresser vers la certification de composants interactifs. Le LII développe djnn [2] un environnement de conception et d'exécution de systèmes interactif.

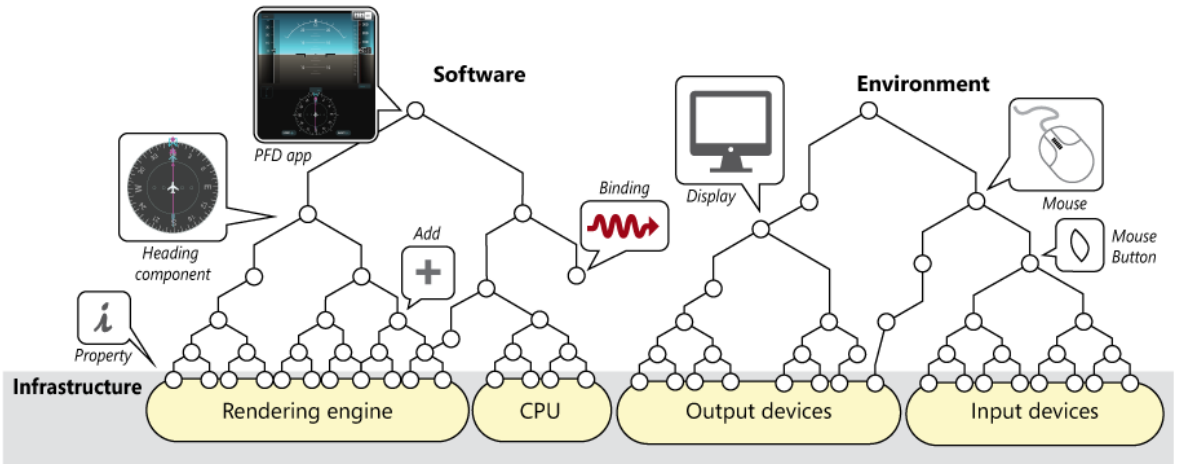

Figure 3.1 – djnn : collection fractale de composants interactifs

Les travaux sur djnn sont motivés à la fois par le besoin des concepteurs de cockpits et de systèmes ATC de disposer d'outils de conception et de validation, et par le souci d'améliorer les modèles théoriques utilisés pour la sécurité des systèmes homme-machine aéronautiques.

3.3 Contexte : les systèmes interactifs de l'aéroport du futur

Les aéroports sont des systèmes complexes. Ils mettent en œuvre une multitude de sous-systèmes de nature très différente avec des objectifs variés. Notamment, ils possèdent souvent une double dimension humain-machine. C'est la symbiose entre ces sous-systèmes mixtes humains et automatiques qui assure la maximisation de la capacité (c-à-d le nombre de départs/arrivés et le nombre de passagers transportés à l'heure) et de la sécurité (pas d'accident) [Gaspard-Boulinc et al., 2013].

L'aéroport du futur emploiera probablement l'automatisation pour augmenter les performances de ces sous-systèmes, en déchargeant certaines opérations des êtres humains pour que ceux-ci se consacrent à des tâches de plus haut-niveau. Ainsi, le développement récent des drones (volant, ou roulant comme les taxibots) impacte fortement la vision de l'aéroport du futur et du contrôle des espaces aériens. Les services rendus par ces véhicules autonomes sont très variés. Au sol, les taxibots électriques et autonomes promettent une amélioration du roulage des avions de transport (réduction des coûts et de l'impact écologique). En l'air les drones peuvent servir pour l'effarouchement aviaire, la surveillance pour le contrôle aérien, ou des missions de sûreté. A terme, des aéronefs sans pilote pourraient également intégrer les espaces aéroportuaires.

2. http://djnn.net

Ce qui caractérise les sous-systèmes de l'aéroport, c'est qu'ils sont en interaction [Arbab, 2006]. Que ce soit entres sous-systèmes automatiques, ou entre être humains, ou entre humains et machines par l'intermédiaire d'interfaces homme-machine, les sous-systèmes interagissent, communiquent, échangent des informations et réagissent aux événements extérieurs prévus et non prévus.

Notre équipe a pour axe de recherche l'ingénierie de l'interaction. Dans le cadre d'autres projets du domaine de l'aéronautique civile, elle développe un environnement dédié à la conception et la programmation des interactions entre humains et systèmes. Par exemple, l'un des problèmes concrets posés par les interactions homme-machine est la combinaison de modalités d'interaction. Il n'existe aucun environnement logiciel fournissant un cadre unifié pour l'ensemble des modalités existantes, des souris et écrans tactiles jusqu'aux capteurs de présence et aux systèmes de suivi du regard et de reconnaissance de la parole. Aussi, intégrer ces modalités dans une même application met souvent en œuvre de multiples langages et méthodes de programmation [Dey et al., 2001b, Baldauf et al., 2007, Henricksen and Indulska, 2006], ce qui accentue notablement la complexité.

L'approche originale initiée par l'ENAC il y a quelques années consiste à accepter la diversité des différentes modalités dans les systèmes interactifs, et à ne pas chercher à leur imposer un cadre unificateur de haut niveau d'abstraction. A l'inverse, la solution retenue consiste à rechercher une unification à un niveau d'abstraction très bas et simple à comprendre : l'échange d'événements. Les modalités sont alors toutes différentes, mais le cadre de base est le même et il est simple à aborder par tous : elles émettent toutes des événements.

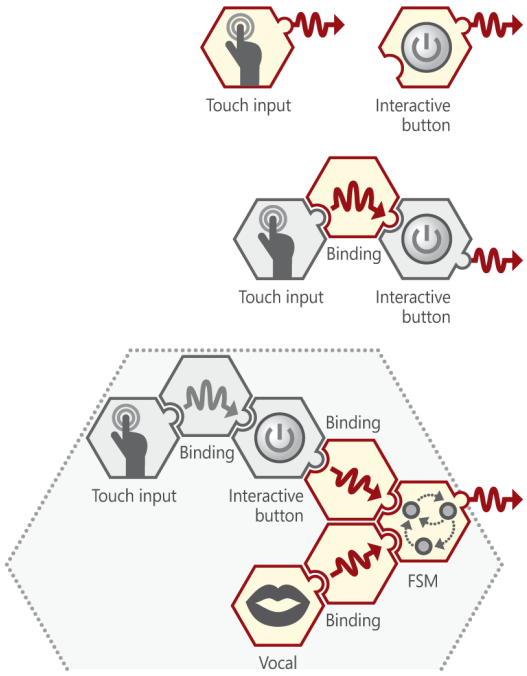

Figure 3.2 – djnn : couplage et activation

Il est possible de prolonger cette approche au sein même des programmes et d'en faire la base d'un nouveau paradigme de programmation et de structuration du code décrivant n'importe quel comportement interactif, y compris entre sous-systèmes automa-

tiques. C'est le programme de recherche ambitieux dans lequel l'ENAC s'est investie depuis quelques années, et sur lequel des résultats significatifs ont été publiés [Chatty et al., 2015, Rey et al., 2015, Magnaudet and Chatty, 2014]. Ces travaux sont concrétisés au sein de l'environnement de programmation djnn, qui a été utilisé dans plusieurs projets en contexte aéronautique [Rey et al., 2015, Conversy et al., 2011, Gaspard-Boulinc et al., 2013].

3.4 Projet de recherche : déploiement de comportements d'objets réactifs

Une première partie du projet consistera à décrire des comportements de véhicule autonome à l'aide des concepts de djnn. Nous voulons notamment décrire les comportements interactifs des taxibots développés dans le cadre du projet MOTA conduit par l'ENAC. Par ailleurs, nous voulons aussi décrire les comportements interactifs des drones Paparazzi développés par le laboratoire Drone de l'ENAC. Il ne s'agira pas de décrire totalement le fonctionnement d'un drone, mais d'en extraire des parties pertinentes pour l'interaction, et de les répliquer avec les concepts de djnn. L'objectif est de fournir une preuve de couverture : il s'agit de s'assurer que les concepts djnn suffisent à décrire n'importe quel comportement réactif d'un taxibot ou d'un drone réel.

Afin de vérifier la correction de la description, il est nécessaire de porter l'environnement d'exécution de djnn sur une plateforme embarquée réaliste. Il faudra donc s'assurer que les algorithmes de l'environnement d'exécution soient compatibles avec les ressources de la plateforme d'exécution. Par ailleurs, il sera nécessaire de développer les interfaces spécifiques avec le monde extérieur : capteurs de vitesse, de luminosité etc. Les deux environnements d'exécution envisagés sont les plateformes paparazzi développées à l'ENAC, et les robots Mindstorm pour simuler les taxibots.

Une fois conçu et programmé, un système développé avec des objets interactifs doit être déployé. S'il est envisageable de réaliser le déploiement de manière manuelle pour un environnement de taille réduite en nombre d'appareils comme c'est le cas dans un cockpit, cela reste une vraie complexité dans le contexte de l'aéroport du futur qui fait également apparaître des problématiques nouvelles.

Le déploiement est un processus complexe qui a pour objectif la mise à disposition puis le maintien en condition opérationnelle d'un système logiciel [Arcangeli et al., 2015, Dearle, 2007a, Carzaniga et al., 1998a]. Les solutions traditionnelles de déploiement, particulièrement dans le monde industriel (RPM, composants MSI, EJB, composants CCM, jusqu'au déploiement de composants pour systèmes virtualisés), privilégient un mode centralisé dans lequel un opérateur humain réalise les différentes tâches du déploiement. Ces solutions répondent principalement aux problèmes du déploiement dans un environnement hiérarchique plus ou moins statique, comme un réseau d'entreprise ou une grille de calcul. Le déploiement est piloté depuis un ordinateur, de manière centralisée, de l'installation à la mise à jour et à la désinstallation. Même dans des travaux de recherche récents [Manzoor and Nefti, 2010], pour déployer, l'opérateur doit posséder une double expertise, sur l'application à déployer et sur les opérations de déploiement. Il lui faut indiquer sur quels appareils les composants du système doivent être déployés, initier le déploiement et être capable de réagir en cas de panne ou d'apparition d'un nouvel appareil.

Pour les systèmes de l'aéroport du futur (interfaces de contrôle, drones, véhicules électriques...) visés par ce projet, l'expression du plan de déploiement ainsi que

la réalisation et la gestion du déploiement sont des tâches humainement impossibles du fait de l'hétérogénéité, de la dynamique, du nombre, mais aussi parce que le domaine de déploiement (l'ensemble des appareils en marche) n'est pas connu à l'avance [Arcangeli et al., 2015]. Ces préoccupations commencent à être considérées [van der Burg and Dolstra, 2014], mais ne permettent pas de prendre en compte ou du moins pas simplement, l'impact du déploiement dynamique sur l'application déployée elle-même car la reconfiguration du déploiement implique la reconfiguration du système réparti. C'est précisément pour répondre à cette problématique qu'il nous semble nécessaire de proposer un cadre d'expression commun au système de déploiement ainsi qu'au système déployé, et dans ce contexte il pourra être construit au moyen de djnn.

Nos précédents travaux [Rottenberg et al., 2014, Boujbel et al., 2014b] nous ont amené à proposer des solutions originales pour le déploiement automatique de systèmes fortement répartis : un processus de déploiement complet ; un langage dédié (DSL) [Boujbel et al., 2014a], qui simplifie la tâche du concepteur du déploiement et permet d'exprimer les propriétés de déploiement ainsi que des informations concernant la perception de l'état du domaine de déploiement ; ainsi qu'un middleware qui assure la génération automatique d'un plan de déploiement en fonction de l'état du domaine, sa réalisation puis le maintien en condition opérationnelle du système.

Toutefois, si ces technologies de déploiement automatique sont un premier pas pour répondre aux besoins identifiés dans le déploiement d'objets réactifs de l'aéroport du futur, il reste à prendre en compte les évolutions dynamiques du domaine de déploiement après le déploiement initial, les besoins de déploiement incrémental (nouvelle spécification de déploiement sur un déploiement existant), et les spécificités liées au déploiement d'objets interactifs [Arcangeli et al., 2015, van Steen et al., 2012]. L'utilisation de djnn pour la réalisation du déploiement devrait également permettre une meilleure intégration entre le middleware de déploiement et les comportements d'objets réactifs déployés.

3.5 Validation : Plateforme Volante

La recherche dans le domaine des systèmes répartis en particulier pose une difficulté, celle de la validation des outils proposés, à cause de la complexité des systèmes réels et du nombre d'entités qui les composent. Une approche souvent utilisée est celle de la simulation [Sulistio et al., 2004]. Malheureusement, si cette technique simplifie la conception, la mise en œuvre et l'évaluation d'une application de démonstration, le réalisme disparaît et la dissémination scientifique liée aux publications est fortement limitée.

Dans notre expérience, le démonstrateur construit pour le projet ANR INCOME (production de middlewares et d'outils de génie logiciel majoritairement) a été basé en partie sur des simulateurs. Par conséquent, il est difficile de communiquer sur le côté applicatif et difficile de convaincre des industriels pour envisager du transfert de technologie. Dans ce projet, nous avons identifié cette difficulté et tenté de varier les moyens de dissémination, par exemple en réalisant une vidéo[3] présentant à la fois les enjeux et les éléments technologiques répondant aux différentes problématiques.

L'ÉNAC, pour permettre d'une part une meilleure dissémination des activités scientifiques ainsi que pour proposer un environnement d'expérimentation et de validation des travaux des différents laboratoires, est entrain de développer des plateformes de recherche

3. https://www.youtube.com/watch?v=Ff4hCdY6OAs

(financement CPER) pour plusieurs segments identifiés dans le concept d'aéroport du futur. En particulier, je m'occupe depuis mi-2016 de concevoir et piloter le projet de plateforme volante (PFV, budget 200k€), qui constitue un des éléments principaux de la composante "bord".

Cette plateforme est pensée comme la dernière étape du processus complet de design d'outils bord : le prototypage et la conception, la validation sur simulateur (simulateur d'intégration), et enfin la validation in-situ. Elle permettra de reproduire certaines conditions non reproductibles en simulateur (contrôle aérien, facteur de charge, turbulences, bruit, vibration, pression temporelle, météo dégradée, luminosité...).

Actuellement, l'appareil envisagé est un TB10 de la flotte de l'ÉNAC, modifié de telle manière que la partie droite du cokpit contenant des instruments redondants et des intruments moteurs, soit simplifiée et intégrée dans la console centrale, afin d'offrir un espace de type "boîte à gants" dans lequel il sera possible d'intégrer des éléments électroniques et informatiques -modules arduino, raspberry pi, tablettes tactiles, éléments de datalink- permettant de développer de nouvelles interactions avec l'appareil ou le sol. Le système est composé de multiples éléments hétérogènes, communiquants, et lui-même connecté aux autes segments (sol, avec une plateforme déployée sur Muret et simulateur, avec un système déployé sur l'ÉNAC), formant un système complexe idéal pour la démonstration de nos travaux (figure 3.3). Pour précision, le processus de certification de ces modifications a été validé par le constructeur de l'appareil (DAHER-SOCATA), ce qui permettra son utilisation en conditions d'exploitation standard, en vol à vue.

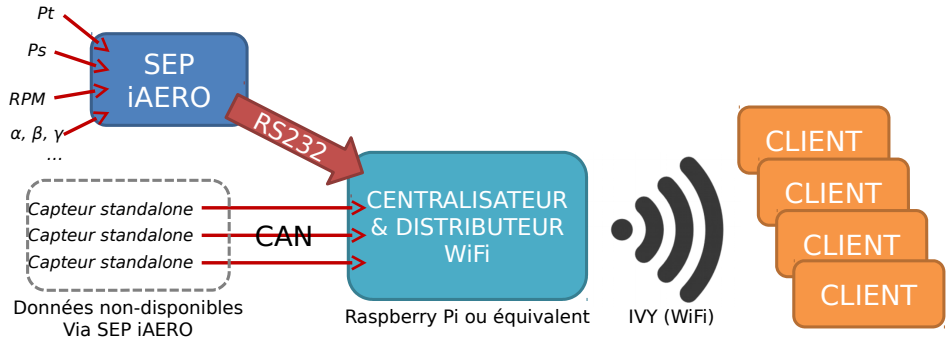

Figure 3.3 – Architecture de la PFV

Des expériences préliminaires ont pu être menées à l'été 2016, avec un double objectif : validation d'un prototype d'interaction ainsi que consolidation des besoins pour la plateforme volante. L'expérimentation a eu lieu sur un vol réel (figure 3.4), dans un appareil de aéro-club de l'ENAC que j'ai piloté, afin de prendre en compte notamment la réalité des turbulences et les variations de luminosité qui ne peuvent être reproduites en simulateur. En parallèle aux résultats obtenus pour le projet ANR AIRTIUS auquel participe mon équipe de recherche, cette expérimentation a permis d'alimenter le cahier des charges de la future plateforme volante. Pour conclure, l'ÉNAC me permet de monter en compétences dans le domaine aéronautique via une formation théorique au brevet de pilote professionnel (CPL-A), afin d'avoir une complémentarité optimale entre mes compétences de chercheur et le domaine applicatif et envisagé pour les prochaines années.

Figure 3.4 – Expérimentation en vol

Bibliographie

[Agha, 1986] Agha, G. (1986). *Actors : a model of concurrent computation in distributed systems*. M.I.T. Press, Cambridge, Ma.

[AMQP, xx] AMQP (xx). AMQP. http://www.amqp.org. Last access in november 2014.

[Androutsellis-Theotokis and Spinellis, 2004] Androutsellis-Theotokis, S. and Spinellis, D. (2004). A survey of peer-to-peer content distribution technologies. *ACM Comput. Surv.*, 36(4) :335–371.

[Arbab, 2006] Arbab, F. (2006). Computing and interaction. In *Interactive Computation*, pages 9–23. Springer.

[Arcangeli et al., 2015] Arcangeli, J.-P., Boujbel, R., and Leriche, S. (2015). Automatic deployment of distributed software systems : Definitions and state of the art. *Journal of Systems and Software*, 103 :198 – 218.

[Arcangeli et al., 2012] Arcangeli, J.-P., Bouzeghoub, A., Camps, V., Canut, M.-F., Chabridon, S., Conan, D., Desprats, T., Laborde, R., Lavinal, E., Leriche, S., Maurel, H., Péninou, A., Taconet, C., and Zaraté, P. (2012). Income – multi-scale context management for the internet of things. In Paternò, F., Ruyter, B., Markopoulos, P., Santoro, C., Loenen, E., and Luyten, K., editors, *Ambient Intelligence*, volume 7683 of *Lecture Notes in Computer Science*, pages 338–347. Springer Berlin Heidelberg.

[Arcangeli et al., 2000] Arcangeli, J.-P., Bray, L., Marcoux, A., Maurel, C., and Migeon, F. (2000). Réflexivité pour la mobilité dans les systèmes d'acteurs. In *4ème École d'Informatique des SYstèmes PArallèles et Répartis (ISPAR'2000)*.

[Arcangeli et al., 2004] Arcangeli, J.-P., Hennebert, V., Leriche, S., Migeon, F., and Pantel, M. (2004). JAVACT 0.5.0 : principes, installation, utilisation et développement d'applications. Technical Report IRIT/2004-5-R, IRIT.

[Arcangeli et al., 2006] Arcangeli, J.-P., Leriche, S., and Pantel, M. (2006). Un framework à composants et agents pour les applications réparties à grande échelle. *Numéro spécial de la revue L'Objet - Agents et Composants*, 4 :102–132.

[Arcangeli et al., 2001] Arcangeli, J.-P., Maurel, C., and Migeon, F. (2001). An API for high-level software engineering of distributed and mobile applications . In *8th IEEE Workshop on Future Trends of Distributed Computing Systems, Bologna (It.)*, pages 155–161, Los Alamitos, Ca., U.S.A. IEEE-CS Press.

[Baldauf et al., 2007] Baldauf, M., Dustdar, S., and Rosenberg, F. (2007). A survey on context-aware systems. *International Journal of Ad Hoc and Ubiquitous Computing*, 2(4) :263–277.

[Blair and Grace, 2012] Blair, G. and Grace, P. (2012). Emergent middleware : Tackling the interoperability problem. *IEEE Internet Computing*, 16(1) :78–82.

[Boinot, 2002] Boinot, P. (2002). *Une approche déclarative de la flexibilité du logiciel*. PhD thesis, Université de Rennes.

[Boujbel et al., 2014a] Boujbel, R., Leriche, S., and Arcangeli, J.-P. (2014a). A framework for autonomic software deployment of multiscale systems. *International Journal on Adanced Systems*, 7(1 & 2) :353–369.

[Boujbel et al., 2014b] Boujbel, R., Rottenberg, S., Leriche, S., Taconet, C., Arcangeli, J.-P., and Lecocq, C. (2014b). MuScADeL : A Deployment DSL based on a Multiscale Characterization Framework. In *COMPSAC Workshops*, pages 708–715. IEEE.

[Bradshaw, 1997] Bradshaw, J. M., editor (1997). *Software agents*. MIT Press, Cambridge, MA, USA.

[Briot, 2004] Briot, J.-P. (2004). Agents et composants : une dualité à explorer. transparents de présentation aux Journées Multi-Agents et Composants, JMAC 2004. http://csl.ensm-douai.fr/MAAC/uploads/3/agents-comp-jmac'04.pdf.

[Briot, 2014] Briot, J.-P. (2014). Composants et agents : évolution de la programmation et analyse comparative. *Technique et Science Informatiques*, 33(1-2) :85–115.

[Broto et al., 2008] Broto, L., Hagimont, D., Stolf, P., Palma, N. D., and Temate, S. (2008). Autonomic management policy specification in Tune. In Wainwright, R. L. and Haddad, H., editors, *23rd Annual Symposium on Applied Computing (SAC 2008)*, pages 1658–1663. ACM.

[Capera et al., 2003] Capera, D., George, J.-P., Gleizes, M.-P., and Glize, P. (2003). The amas theory for complex problem solving based on self-organizing cooperative agents. In *Twelfth International Workshop on Enabling Technologies : Infrastructure for Collaborative Enterprises*.

[Carzaniga et al., 1998a] Carzaniga, A., Fuggetta, A., Hall, R. S., Heimbigner, D., van der Hoek, A., and Wolf, A. L. (1998a). A characterization framework for software deployment technologies. Technical report, Defense Technical Information Center (DTIC) Document.

[Carzaniga et al., 1998b] Carzaniga, A., Fuggetta, A., Hall, R. S., Heimbigner, D., van der Hoek, A., Wolf, A. L., Der, A. V., Wolf, E. L., and Wolf, E. L. (1998b). A characterization framework for software deployment technologies. Technical report, Dept. of Computer Science, University of Colorado.

[Carzaniga et al., 1998c] Carzaniga, A., Fuggetta, A., Hall, R. S., van der Hoek, A., Heimbigner, D., and Wolf, A. L. (1998c). A characterization framework for software deployment technologies. Technical Report CU-CS-857-98, Dept. of Computer Science, University of Colorado.

[Chatty et al., 2015] Chatty, S., Magnaudet, M., and Prun, D. (2015). Verification of properties of interactive components from their executable code. In *Proceedings of the 7th ACM SIGCHI Symposium on Engineering Interactive Computing Systems*, pages 276–285. ACM.

[Chavalarias et al., 2009] Chavalarias et al., D. (2009). French roadmap for complex systems 2008-2009.

[Chen, 2012] Chen, Y.-K. (2012). Challenges and opportunities of internet of things. In *Design Automation Conference (ASP-DAC), 2012 17th Asia and South Pacific*, pages 383 –388.

[C.H.O.C.O. Team, 2010] C.H.O.C.O. Team (2010). CHOCO : an open source Java constraint programming library. Technical Report 10-02-INFO, Ecole des Mines de Nantes. Last access : June 2014.

[Conversy et al., 2011] Conversy, S., Gaspard-Boulinc, H., Chatty, S., Valès, S., Dupré, C., and Ollagnon, C. (2011). Supporting air traffic control collaboration with a tabletop system. In *Proceedings of the ACM 2011 conference on Computer supported cooperative work*, pages 425–434. ACM.

[da Silva e Silva et al., 2003] da Silva e Silva, F., Endler, M., and Kon, F. (2003). Developing adaptive distributed applications : a framework overview and experimental results. In *International symposium on Distributed Objects and Application, Catania, Sicile*.

[Dearle, 2007a] Dearle, A. (2007a). Software Deployment, Past, Present and Future. In Briand, L. C. and Wolf, A. L., editors, *Workshop on the Future of Software Engineering (FOSE 2007)*, pages 269–284.

[Dearle, 2007b] Dearle, A. (2007b). Software deployment, past, present and future. In *FOSE*, pages 269–284.

[Dearle et al., 2010] Dearle, A., Kirby, G. N. C., and McCarthy, A. (2010). A framework for constraint-based deployment and autonomic management of distributed applications. *CoRR*, abs/1006.4572.

[Dearle et al., 2004] Dearle, A., Kirby, G. N. C., and McCarthy, A. J. (2004). A framework for constraint-based deployment and autonomic management of distributed applications. In *ICAC*.

[DECOR 04, 2004] DECOR 04 (2004). *DECOR'2004, Actes de la 1re Conférence Francophone sur le Déploiement et la (Re)Configuration de Logiciels*. ISBN-2-7261-1276-5.

[Dey et al., 2001a] Dey, A., Abowd, G., and Salber, D. (2001a). A Conceptual Framework and a Toolkit for Supporting the Rapid Prototyping of Context-Aware Applications. *Special Issue on Context-Aware Computing in the Human-Computer Interaction Journal*, 16(2–4) :97–166.

[Dey et al., 2001b] Dey, A. K., Abowd, G. D., and Salber, D. (2001b). A conceptual framework and a toolkit for supporting the rapid prototyping of context-aware applications. *Human-computer interaction*, 16(2) :97–166.

[Dikaiakos et al., 2009] Dikaiakos, M. D., Katsaros, D., Mehra, P., Pallis, G., and Vakali, A. (2009). Cloud Computing : Distributed Internet Computing for IT and Scientific Research. *IEEE Internet Computing*, 13(5) :10–13.

[Dowling and Cahill, 2001] Dowling, J. and Cahill, V. (2001). *The K-Component Architecture Meta-Model for Self-Adaptive Software*, pages 81–88. Springer Berlin Heidelberg, Berlin, Heidelberg.

[Déjean, 2003] Déjean, T. (2003). Développement d'applications multi-agents adaptatifs avec l'api javact. Technical report, Institut de Recherche en Informatique de Toulouse.

[Felix, xx] Felix (xx). Apache Felix - Welcome to Apache Felix. `https://felix.apache.org`. Last access in november 2014.

[Ferber, 1995] Ferber, J. (1995). *Les systèmes multi-agents - Vers une intelligence collective*. InterEditions.

[Filho et al., 2000] Filho, R. S. S., Silveira, R., and Filho, S. (2000). Mobile agents and software deployment.

[Flissi et al., 2008a] Flissi, A., Dubus, J., Dolet, N., and Merle, P. (2008a). Deploying on the grid with deployware. In *CCGRID*, pages 177–184. IEEE Computer Society.

[Flissi et al., 2008b] Flissi, A., Dubus, J., Dolet, N., and Merle, P. (2008b). Deploying on the grid with deployware. In *CCGRID*, pages 177–184.

[Forum, 2008] Forum, U. (2008). Upnp device architecture.

[Foster and Kesselman, 1998] Foster, I. and Kesselman, C. (1998). *The Grid : Blueprint for a New Computing Infrastructure*. Morgan Kaufmann, San Francisco, California.

[Foster et al., 2001] Foster, I., Kesselman, C., and Tuecke, S. (2001). The anatomy of the grid : Enabling scalable virtual organizations. *International journal of high performance computing applications*, 15(3) :200–222.

[Franklin et al., 2005] Franklin, M. J., Jeffery, S. R., Krishnamurthy, S., and Reiss, F. (2005). Design Considerations for High Fan-in Systems : The HiFi Approach. In *In CIDR,Conference on Innovative Data Systems Research*, pages 290–304, Asilomar, Californie.

[Gamma et al., 1995] Gamma, E., Helm, R., Johnson, R., and Vlissides, J. (1995). *Design patterns : Elements of Reusable Object-Oriented Software*. Addison-Wesley Professional Computing Series.

[Garlan and Shaw, 1994] Garlan, D. and Shaw, M. (1994). An introduction to software architecture. Technical report, Pittsburgh, PA, USA.

[Gaspard-Boulinc et al., 2013] Gaspard-Boulinc, H., Chatty, S., Hurter, C., Marcy, J., Conversy, S., and Lesbordes, R. (2013). Collaboration et tangibilité : de nouvelles perspectives pour le contrôle aérien. In *IHM 2013, 25ème conférence francophone sur l'Interaction Homme-Machine*.

[Goldsack et al., 2009] Goldsack, P., Guijarro, J., Loughran, S., Coles, A. N., Farrell, A., Lain, A., Murray, P., and Toft, P. (2009). The SmartFrog configuration management framework. *Operating Systems Review*, 43(1) :16–25.

[Group, 2006] Group, O. M. (2006). Corba component model 4.0 specification. Specification Version 4.0, Object Management Group.

[Gson, xx] Gson (xx). google-gson - A Java library to convert JSON to Java objects and vice-versa - Google Project Hosting. `http://code.google.com/p/google-gson/`. Last access in november 2014.

[Gubbi et al., 2013] Gubbi, J., Buyya, R., Marusic, S., and Palaniswami, M. (2013). Internet of Things (IoT) : A vision, architectural elements, and future directions. *Future Generation Computer Systems*, 29(7) :1645 – 1660.

[Hall et al., 1997] Hall, R., Heimbigner, D., van der Hoek, A., and Wolf, A. (1997). The software dock : A distributed, agent-based software deployment system. Technical Report CU-CS-832-97, Department of Computer Science, University of Colorado.

[Hall et al., 1999] Hall, R. S., Heimbigner, D., and Wolf, A. L. (1999). A cooperative approach to support software deployment using the software dock. In *ICSE '99*.

[Harrison et al., 1995] Harrison, C. G., Harrison, C. G., Chess, D. M., Chess, D. M., Kershenbaum, A., and Kershenbaum, A. (1995). Mobile agents : Are they a good idea ?

[Henricksen and Indulska, 2006] Henricksen, K. and Indulska, J. (2006). Developing context-aware pervasive computing applications : Models and approach. *Pervasive and mobile computing*, 2(1) :37–64.

[Hewitt, 1977] Hewitt, C. (1977). Viewing control structures as patterns of passing messages. *Artificial Intelligence*, 8(3) :323–364.

[ISO 9000:2005, 2009] ISO 9000:2005 (2009). Quality management systems – Fundamentals and vocabulary.

[ISO/IEC/IEEE, 2011] ISO/IEC/IEEE (2011). Systems and software engineering — Architecture description. International Standard ISO/IEC/IEEE-42010 :2011, ISO/IEC/IEEE Joint Technical Committee.

[Jetty, xx] Jetty (xx). Apache Felix - Apache Felix HTTP Service. http://felix.apache.org/documentation/subprojects/apache-felix-http-service.html. Last access in november 2014.

[jOpt, xx] jOpt (xx). jOpt, Java OPL implementation. http://jopt.sourceforge.net/opl.php. Last access in november 2014.

[JSON, xx] JSON (xx). JSON. http://www.json.org. Last access in november 2014.

[Juhász et al., 2004] Juhász, Z., Kacsuk, P., and Kranzlmuller, D. (2004). *Distributed and Parallel Systems : cluster and grid computing*, volume 777. Springer Science & Business Media.

[Kessis et al., 2009] Kessis, M., Roncancio, C., and Lefebvre, A. (2009). DASIMA : A flexible management middleware in multi-scale contexts. In *6th Int. Conf. on Information Technology : New Generations (ITNG '09)*, pages 1390–1396.

[Kuchcinski and Szymanek, xx] Kuchcinski, K. and Szymanek, R. (xx). JaCoP - Java constraint programming solver. http://jacop.osolpro.com. Last access in november 2014.

[Leriche and Arcangeli, 2004] Leriche, S. and Arcangeli, J.-P. (2004). Une architecture pour les agents mobiles adaptables. In *Actes des Journées Composants JC'2004*.

[Leriche and Arcangeli, 2005] Leriche, S. and Arcangeli, J.-P. (2005). Apports mutuels des technologies P2P, agents mobiles et composants logiciels. In *Actes du workshop OCM-SI'05*.

[Leriche and Arcangeli, 2006] Leriche, S. and Arcangeli, J.-P. (2006). Vers un modèle d'agent flexible. In *Actes des Journées Multi-Agent et Composant JMAC'2006*.

[Leriche and Arcangeli, 2010] Leriche, S. and Arcangeli, J.-P. (2010). Flexible architectures of adaptive agents : the agentϕ approach. *International journal of grid computing and multi agent systems (IJGCMAS)*, 1(1) :55–75. 8878.

[Leriche et al., 2004] Leriche, S., Arcangeli, J.-P., and Pantel, M. (2004). Agents mobiles adaptables pour les systèmes d'information pair à pair hétérogènes et répartis . In *Nouvelles Technologies de la Répartition, NOTERE 2004 , Saidia (Ma.)*, pages 29–43. CIISE - ENST - Univ. Mohammed 1er.

[Louberry et al., 2011a] Louberry, C., Roose, P., and Dalmau, M. (2011a). Kalimucho : Adaptation Platform for Mobile Applications. In *11th Int. Conf. on New Technologies of Distributed Systems (NOTERE 2011)*, pages 1–8.

[Louberry et al., 2011b] Louberry, C., Roose, P., and Dalmau, M. (2011b). Kalimucho : Contextual Deployment for QoS Management. In *11th IFIP WG 6.1 International Conference, DAIS 2011, Reykjavik, Iceland, June 2011, Proceedings*, volume 6723 of *Lecture Notes in Computer Science*, pages pp.43–56. Springer.

[Magnaudet and Chatty, 2014] Magnaudet, M. and Chatty, S. (2014). What should adaptivity mean to interactive software programmers ? In *Proceedings of the 2014 ACM SIGCHI symposium on Engineering interactive computing systems*, pages 13–22. ACM.

[Manzoor and Nefti, 2010] Manzoor, U. and Nefti, S. (2010). QUIET : A Methodology for Autonomous Software Deployment using Mobile Agents. *J. Network and Computer Applications*, 33(6) :696–706.

[Marcoux et al., 1998] Marcoux, A., Maurel, C., Migeon, F., and Sallé, P. (1998). Generic operational decomposition for concurrent systems : semantics and reflection. *Parallel and Distributed Computing Practices*, 1(4) :49–64.

[Matougui and Leriche, 2011] Matougui, M. E. A. and Leriche, S. (2011). Vers un environnement de déploiement autonomique. In *Ubimob'2011 : Septièmes journées francophones Mobilité et Ubiquité*, pages 57–62.

[Mattern, 2001] Mattern, F. (2001). The vision and technical foundations of ubiquitous computing. *Upgrade*, 2(5) :2–6.

[McAffer, 1995] McAffer, J. (1995). Meta-level programming with coda. In *Proceedings of the European Conference on Object-Oriented Computing (ECOOP), LNCS 952*, pages 190–214. Springer-Verlag.

[Medvidovic, 1996] Medvidovic, N. (1996). Adls and dynamic architecture changes. In *Proceedings of the Second International Software Architecture Workshop (ISAW-2)*, pages 24–27.

[Mell and Grance, 2011] Mell, P. and Grance, T. (2011). The NIST Definition of Cloud Computing. NIST Special Publication 800-145, National Institute of Standards and Technology, Gaithersburg, USA.

[Milojicic et al., 2002] Milojicic, D., Kalogeraki, V., Lukose, R., Nagaraja, K., Pruyne, J., Richard, B., Rollins, S., and Xu, Z. (2002). Peer-to-peer computing. Technical Report HPL-2002-57, HP Laboratories Palo Alto.

[Miorandi et al., 2012] Miorandi, D., Sicari, S., De Pellegrini, F., and Chlamtac, I. (2012). Internet of things : Vision, applications and research challenges. *Ad Hoc Networks*, 10(7) :1497–1516.

[OMG, 2008a] OMG (2008a). MOF Model to Text Transformation Language, v1.0. formal/2008-01-16. Object Management Group.

[OMG, 2008b] OMG (2008b). Software & Systems Process Engineering Metamodel (SPEM), v2.0. formal/2008-04-01. Object Management Group.

[ortools, xx] ortools (xx). or-tools, operations research tools developed at Google. `https://code.google.com/p/or-tools`. Last access in november 2014.

[Oussalah, 2005] Oussalah, M. (2005). *Ingenierie des Composants : Concepts, techniques et outils*. Vuibert Informatique. Ouvrage collectif.

[RabbitMQ, xx] RabbitMQ (xx). RabbitMQ : Messaging that just work. `http://www.rabbitmq.com`. Last access in november 2014.

[Rao and Georgeff, 1995] Rao, A. S. and Georgeff, M. P. (1995). BDI-agents : from theory to practice. In *Proceedings of the First Intl. Conference on Multiagent Systems*, San Francisco.

[Ratnasamy et al., 2001] Ratnasamy, S., Francis, P., Handley, M., Karp, R., and Shenker, S. (2001). A scalable content-addressable network. In *Proceedings of the ACM SIGCOMM '01 Conference*, pages 161–172.

[REST, xx] REST (xx). REST. `http://www.xfront.com/REST-Web-Services.html`. Last access in november 2014.

[Rey et al., 2015] Rey, S., Conversy, S., Magnaudet, M., Poirier, M., Prun, D., Vinot, J.-L., and Chatty, S. (2015). Using the djnn framework to create and validate interactive components iteratively. In *Proceedings of the 7th ACM SIGCHI Symposium on Engineering Interactive Computing Systems*, pages 230–233. ACM.

[Rottenberg et al., 2014] Rottenberg, S., Leriche, S., Taconet, C., Lecocq, C., and Desprats, T. (2014). MuSCa : A multiscale characterization framework for complex distributed systems. In Ganzha, M., Maciaszek, L. A., and Paprzycki, M., editors, *Proceedings of the 2014 Federated Conference on Computer Science and Information Systems, Warsaw, Poland, September 7-10, 2014.*, pages 1657–1665.

[Sabharwal, 2006] Sabharwal, R. (2006). Grid Infrastructure Deployment using SmartFrog Technology. In *Int. Conf. on Networking and Services (ICNS 2006)*, page 73. IEEE Computer Society.

[Saint-Andre et al., 2009] Saint-Andre, P., Smith, K., and Tronçon, R. (2009). *XMPP : The Definitive Guide : Building Real-Time Applications with Jabber Technologies*. O'Reilly Media, Inc.

[Satyanarayanan, 2011] Satyanarayanan, M. (2011). Mobile computing : the next decade. *Mobile Computing and Communications Review*, 15(2) :2–10.

[Satyanarayanan et al., 2009] Satyanarayanan, M., Bahl, P., Cáceres, R., and Davies, N. (2009). The Case for VM-Based Cloudlets in Mobile Computing. *IEEE Pervasive Computing*, 8(4) :14–23.

[SPEM2.0, 2008] SPEM2.0 (2008). SPEM 2.0. http://www.omg.org/spec/SPEM/2.0/. Last access in november 2014.

[Stoica et al., 2001] Stoica, I., Morris, R., Karger, D., Kaashoek, M. F., and Balakrishnan, H. (2001). Chord : A scalable peer-to-peer lookup service for internet applications. In *Proceedings of the ACM SIGCOMM '01 Conference*, pages 149–160.

[Strembeck and Zdun, 2009] Strembeck, M. and Zdun, U. (2009). An approach for the systematic development of domain-specific languages. *Software : Practice and Experience*, 39(15) :1253–1292.

[Sulistio et al., 2004] Sulistio, A., Yeo, C. S., and Buyya, R. (2004). A taxonomy of computer-based simulations and its mapping to parallel and distributed systems simulation tools. *Software-Practice and Experience*, 34(7) :653–674.

[Szyperski, 2002] Szyperski, C. (2002). *Component Software - Beyond Object-Oriented Programming*. Addison-Wesley / ACM Press.

[Tamura, 2003] Tamura, N. (2003). Cream : Class library for constraint programming in java. http://bach.istc.kobe-u.ac.jp/cream/.

[Tamura, xx] Tamura, N. (xx). Copris : Constraint Programming in Scala. http://bach.istc.kobe-u.ac.jp/copris. Last access in november 2014.

[Tolvanen and Kelly, 2010] Tolvanen, J.-P. and Kelly, S. (2010). Integrating models with domain-specific modeling languages. In *Proceedings of the 10th Workshop on Domain-Specific Modeling*, DSM '10, pages 10–1, New York, NY, USA. ACM.

[Toure et al., 2010] Toure, M., Stolf, P., Hagimont, D., and Broto, L. (2010). Large scale deployment. In *6th Int. Conf. on Autonomic and Autonomous Systems (ICAS)*, pages 78–83. IEEE Computer Society.

[van der Burg and Dolstra, 2011] van der Burg, S. and Dolstra, E. (2011). A Self-Adaptive Deployment Framework for Service-Oriented Systems. In Giese, H. and Cheng, B. H. C., editors, *6th International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS'11)*, pages 208–217. ACM.

[van der Burg and Dolstra, 2014] van der Burg, S. and Dolstra, E. (2014). Disnix : A Toolset for Distributed Deployment. *Science of Computer Programming*, 79 :52–69.

[Van Deursen et al., 2000] Van Deursen, A., Klint, P., and Visser, J. (2000). Domain-specific languages : An annotated bibliography. *ACM Sigplan Notices*, 35(6) :26–36.

[van Steen et al., 2012] van Steen, M., Pierre, G., and Voulgaris, S. (2012). Challenges in very large distributed systems. *Journal of Internet Services and Applications*, 3 :59–66. 10.1007/s13174-011-0043-x.

[Waldner, 2007] Waldner, J. (2007). *Nano-informatique et intelligence ambiante : inventer l'ordinateur du XXIe siècle*. Hermès science publications.

[Weiser, 1991] Weiser, M. (1991). The computer for the 21st century. *Scientific American*.

[Weiser, 1996] Weiser, M. (1996). Nomadic Issues in Ubiquitous Computing. http://www.ubiq.com/hypertext/weiser/NomadicInteractive/. Slides presented at the NOMADIC'96 Conference, last access in november 2014.

[Xtend, xx] Xtend (xx). Xtend - Modernized Java. http://www.eclipse.org/xtend. Last access in november 2014.

[Xtext, xx] Xtext (xx). Xtext - Language Development made Easy ! http://www.eclipse.org/Xtext/. Last access in november 2014.

Annexes

A Curriculum Vitae

A.1 Parcours Professionel

Date	Fonction	Etablissement-Lieu
Depuis 2013	**Enseignant-Chercheur** Laboratoire d'Informatique Interactive Axe ingéniérie	ENAC / Ecole Nationale de l'Aviation Civile, Toulouse
2007-2013	**Enseignant-Chercheur** Département informatique Equipe MARGE	Télécom SudParis (ex. INT), CNRS UMR 5157, Evry
2006-2007	**ATER** (mi-temps)	Université Paul Sabatier/ IRIT, Toulouse

A.2 Parcours Universitaire

Date	Diplôme	Etablissement-Lieu
2007 & 2012	**Qualification** aux fonctions de Maître de Conférences, section 27	
2003-2006	**Allocataire de recherche / Moniteur** Thèse de doctorat sous la direction de J.-P Arcangeli Bourse ministérielle Mention « très honorable »	Université Paul Sabatier / IRIT, Toulouse
1998-2003	**Cursus universitaire complet** (en 5 ans) DEA d'Informatique Fondamentale, mention B, bourse de mérite universitaire Maîtrise d'Informatique, mention AB Licence d'Informatique, mention B DEUG MIAS, options informatique et astrophysique	Université Paul Sabatier / IRIT, Toulouse

B Liste des publications

B.1. Revues internationales indexées

1. Arcangeli J., Boujbel R., Leriche S. 2015. *Automatic Deployment of Distributed Software Systems : Definitions and State of the Art.* In Journal of Systems and Software.
2. Boujbel R., Leriche S., Arcangeli J. 2014. *MuScADeL, a DSL for Multiscale and Autonomic Software Deployment*. In International Journal On Advances in Software (IJAS),.
3. Sam Rottenberg, Claire Lecocq, Sébastien Leriche. Design and Evaluation of a Project-Based Learning Ubiquitous Platform for Universal Client : PBL2U. In : International Journal of Mobile and Blended Learning. 2012
4. Sébastien Leriche and Jean-Paul Arcangeli. Flexible Architectures of Adaptive Agents : the Agentφ approach. In : International Journal of Grid Computing and Multi-Agent Systems. 2010
5. Sébastien Leriche and Jean-Paul Arcangeli. Agentφ : A Tool for Modeling Composite Self-Adaptive Agents. In : International Transactions on Systems Science and Applications. 2008.

B.2. Ouvrages, contributions à ouvrages ou chapitres

1. Hugues D., Taconet C., Leriche S. 2015. Proceedings of the 2nd ACM workshop on middleware for context-aware applications in the IoT : In conjunction with ACM/IFIP/USENIX ACM international middleware conference : December 8 2015 : Vancouver, Canada.
2. Hughes D., Taconet C., Leriche S. 2014. *M4IOT'14 : Proceedings of the 1st ACM Workshop on Middleware for Context-Aware Applications in the IoT*

B.3. Actes de colloques et congrès nationaux et internationaux à comité de lecture

Conférences et workshops internationaux

1. Rottenberg S., Leriche S., Taconet C., Lecocq C., DESPRATS T. 2014. *MuSCa : a multiscale characterization framework for complex distributed systems*. In FEDCSIS 2014
2. Boujbel R., Rottenberg S., Leriche S., Taconet C., Arcangeli J., Lecocq C. 2014. *MuScADeL : A Deployment DSL based on a Multiscale Characterization Framework*. In COMPSAC (2014)
3. Rottenberg S., Leriche S., Taconet C., Lecocq C., Desprats T. 2014. *MuSCa : A Multiscale Characterization Framework for Complex Distributed Systems*. FedCSIS/IEEE 2014.
4. Boujbel R., Arcangeli J., Leriche S. 2013. *A DSL for Multi-Scale and Autonomic Software Deployment*. ICSEA 2013
5. MATOUGUI Mohamed El Amine, LERICHE Sébastien. *A middleware architecture for autonomic software deployment.* ICSNC '12 : The Seventh International Conference on Systems and Networks Communications , XPS, 18-23 november 2012
6. Mohamed El Amine Matougui, Sébastien Leriche. *j-ASD : un middleware pour le déploiement logiciel autonomique*. NOTERE/CFIP '12 : Conférence Internationale Nouvelles Technologies de la Répartition, 2012.

7. ROTTENBERG Sam, LECOCQ Claire, LERICHE Sébastien. PBL2U : A PROJECT-BASED LEARNING UBIQUITOUS PLATFORM FOR UNIVERSAL CLIENT. Dans / In :Mobile Learning 2011
8. MATOUGUI Mohamed El Amine, LERICHE Sébastien. Validation of COSMOS DSL programs. Dans / In : ICCES '10
9. MATOUGUI Mohamed El Amine, LERICHE Sébastien. *Validation of COSMOS DSL programs.* ICCES 2010 : 6th International Conference on Computer Engineering & Systems, IEEE, 30 november - 02 december 2010, Cairo, Egypt, 2010
10. Judicael Ribault, Olivier Dalle, Denis Conan and Sebastien Leriche. OSIF : A Framework To Instrument, Validate, and Analyze Simulations. In : SIMUTools 2010
11. Jean-Paul Arcangeli, Sébastien Leriche. Construction d'agents auto-adaptatifs à base de micro-composants opératoires. Dans / In : Journées Francophones des Systèmes Multi-Agents (JFSMA 2007)
12. Sébastien Leriche, Jean-Paul Arcangeli. Modeling Self-Adaptive Agents for Ubiquitous and Pervasive Computing. Dans / In : International Workshop on Multi-Agent Systems Challenges for Ubiquitous and Pervasive Computing, 2007.
13. Sébastien Leriche, Jean-Paul Arcangeli. Adaptive Autonomous Agent Models for Open Distributed Systems. Dans / In : International Multi-Conference on Computing in the Global Information Technology (ICCGI 2007)
14. Sébastien Leriche, Jean-Paul Arcangeli. Flexible Architectures and Agents for Adaptive Autonomic Systems. Dans / In : IEEE International Workshop on Engineering of Autonomic and Autonomous Systems (EASe 2007),
15. Jean-Paul Arcangeli, Sébastien Leriche, Marc Pantel. Development of Flexible Peer-to-Peer Information Systems using Adaptable Mobile Agents. Dans / In : 1st Int. Workshop on Grid and Peer-to-Peer Computing Impacts on Large Scale Heterogeneous Distributed Database Systems (GLOBE'04)
16. Sébastien Leriche, Jean-Paul Arcangeli, Marc Pantel. Agents mobiles adaptables pour les systèmes d'information pair à pair hétérogènes et répartis. Dans / In : Nouvelles Technologies de la Répartition, NOTERE 2004

Conférences et workshops nationaux

17. Rottenberg S., Leriche S., Taconet C., Lecocq C., DESPRATS T. 2014. *MuScA : a multiscale distributed systems scale-awareness framework*. In CAL 2014 : Conférence francophone sur les architectures logicielles, 1-14
18. Boujbel R., Leriche S., Arcangeli J. 2014. *Formalisation de l'expression d'un plan de déploiement autonomique à base de contraintes*. In UbiMob2014 : 10èmes journées francophones Mobilité et Ubiquité.
19. Arcangeli J., Chabridon S., Conan D., Desprats T., Laborde R., Leriche S., Lim L., Taconet C., Boujbel R., Machara Marquez S., Marie P., Rottenberg S. 2014. *Gestion de contexte multi-échelle pour l'Internet des objets (regular paper)*. In Journées francophones Mobilité et Ubiquité (UBIMOB)
20. Arcangeli J., Bouzeghoub A., Camps V., Chabridon S., Conan D., Desprats T., Laborde R., Lavinal E., Leriche S., Maurel H., Mbarki M., Péninou A., Taconet C., Zaraté P., Boujbel R., Lim L., Machara Marquez S., Marie P., Mignard C., Oglaza A., Rottenberg S. 2014. *Projet INCOME : INfrastructure de gestion de COntexte Multi-Echelle pour*

l'Internet des Objets (short paper). In Conférence Francophone sur les Architectures Logicielles (CAL),

21. Boujbel R., Leriche S., Arcangeli J., Kem O. 2014. *Formalisation de l'expression d'un plan de déploiement autonomique à base de contraintes*. In Journées francophones Mobilité et Ubiquité - UBIMOB 2014
22. MATOUGUI Mohamed El Amine, LERICHE Sébastien. *Vers un environnement de déploiement autonomique*. Ubimob 2011 : 7es journées francophones Mobilité et ubiquité, IRIT, 06-08 juin 2011, Toulouse, France, 2011
23. MATOUGUI Mohamed El Amine, LERICHE Sébastien. Validation de programmes Cosmos DSL. Dans / In : MajecSTIC 2010
24. Sébastien Leriche, Jean-Paul Arcangeli. Vers un modèle d'agent flexible. Dans / In : Journée Multi-Agent et Composant (JMAC 2006)
25. Sébastien Leriche, Jean-Paul Arcangeli. Apport mutuels des technologies pair à pair, agents mobiles et composants logiciels. Dans / In : 4ème Workshop sur les Objets, Composants et Modèles dans l'Ingénierie des Systèmes d'Information
26. Sébastien Leriche, Jean-Paul Arcangeli, Christelle Chaudet. Adaptation et déploiement d'applications réparties ouvertes. Dans / In : Journées Composants 2005
27. Sébastien Leriche, Jean-Paul Arcangeli. Déploiement et adaptation de systèmes P2P : une approche à base d'agents mobiles et de composants. Dans / In : Journées Multi-Agents et Composants, JMAC 2004
28. Jean-Paul Arcangeli, Sébastien Leriche, Marc Pantel. Déploiement de composants de recherche d'information par des agents mobiles. Dans / In : 3ème Workshop sur les Objets, Composants et Modèles dans l'Ingénierie des Systèmes d'Information
29. Sébastien Leriche, Jean-Paul Arcangeli. Une architecture pour les agents mobiles adaptables. Dans / In : Journées Composants JC'04

B.4. Autres contributions significatives

Démonstrations

Kem O., Boujbel R., Leriche S. 2014. Récupération de l'état d'un domaine de déploiement (démonstration). Journées francophones Mobilité et Ubiquité - UBIMOB 2014, 5 June 2014 - 6 June 2014

Rapports

Jean-Paul Arcangeli , Amel Bouzeghoub , Valérie Camps , Sophie Chabridon , Denis Conan , Thierry Desprats , Romain Laborde , Sébastien Leriche , Hervé Maurel , Mohamed Mbarki , André Péninou , Chantal Taconet , Pascale Zaraté. Méthodes et outils intergiciels de gestion de contexte multi-échelle pour l'Internet des objets : programme scientifique du projet ANR INCOME. IRIT/RR—2014–11--FR, 2014

Thèse

Sébastien Leriche. Architectures à composants et agents pour la conception d'applications réparties adaptables. Thèse de doctorat, Université Paul Sabatier, décembre 2006.

C Activités d'encadrement

Les 3 thèses co-encadrées sont dans le domaine informatique de la section CNU 27.

C.1 Etudiants en thèse

Nom	Amine MATOUGUI
Sujet	Vers un environement pour le déploiement logiciel autonomique
Date de soutenance	21 novembre 2013
Taux d'encadrement	90 %
Liste des travaux co-publiés	**Références « conférences » 5, 6, 8, 9, 22, 23**
Situation actuelle du docteur	Ingénieur étude et développement C#, .NET, Web – CGI, Paris

Le financement de la thèse d'A. MATOUGUI a été obtenu sur une bourse « Institut Carnot », sur un projet scientifique que j'ai préparé et présenté (100%). Le premier directeur de recherche a été Guy BERNARD, remplacé par Chantal TACONET à son départ à la retraite. Guy et Chantal m'ont laissé une totale liberté d'encadrement.

Nom	Sam ROTTENBERG
Sujet	Modèles, méthodes et outils pour les systèmes répartis multiéchelles
Date de soutenance	27 avril 2015
Taux d'encadrement	40 %
Liste des travaux co-publiés	**Références « revues » 3** **Références « conférences » 1, 2, 3, 7, 17, 19, 20**
Situation actuelle du docteur	Software Architect chez Zengularity, Paris

Sam ROTTENBERG a été encadré par Chantal TACONET (qui a été la directrice de recherche) à part égale avec moi et Claire LECOCQ. Le financement a été acquis sur une bourse de l'institut Télécom (proposition conjointe des 3 encadrants).

Nom	Raja BOUJBEL
Sujet	Déploiement de systèmes répartis multi-échelles : processus, langage et outils intergiciels
Date de soutenance	30 janvier 2015
Taux d'encadrement	50 %
Liste des travaux co-publiés	**Références « revues » 1, 2** **Références « conférences » 2, 4, 18, 19, 20, 21**
Situation actuelle du docteur	Ingénieur de recherche – SOGETI, Toulouse

Le financement de la thèse R. Boujbel a été obtenue sur un projet ANR (INCOME 2012-2016, programme INFRA) accepté en 2011, et dont la tâche (n°5) nécessitant le recrutement d'un doctorant a été complètement spécifié par mes soins. Jean-Paul Arcangeli, responsable du projet et de la tâche 5 a co-dirigé cette thèse avec moi.

C.2 Etudiants de Master 2

Nom	**Sujet**	**Période**
Amine MATOUGUI	Validation de programmes DSL-COMOS et intégration dans un plugin Eclipse	2009 (2d semestre)
Oudom KEM	Infrastructure pour le déploiement autonomique	2013 (2d semestre)
Wang YUE	Expérimentations de déploiement logiciel	2012 (2s semestre)

D Activités d'enseignement

D.1 Principaux thèmes

Programmation (C, Java) L3 (bases algo, objet), M1 (programmation avancée), M2 (systèmes répartis, mobilité)

Modélisation (UML, DSL) M1 (objet et système), M2 (déploiement logiciel)

Sécurité (théorie/concepts, applications web, serveurs Java, virus/antivirus) en M1 et M2 en particulier interventions hors ENAC dans les master SECURITE d'Evry et TLSSEC (INP)

Ingénierie système (concepts et applications) L3 (processus, V&V) et M1 (processus appliqués à l'aéronautique)

D.2 Responsabilités collectives

Depuis 2015 à l'ENAC, je suis **responsable de formation** d'un parcours de niveau M1 « mineure SITA » qui regroupe des enseignements de conception/-modélisation/programmation orientée objet et d'intelligence artificielle appliquée aux systèmes de contrôle du trafic aérien (12 intervenants, 5 modules).

Entre 2009 et 2011 à TSP, j'ai eu la **co-responsabilité** d'un module d'enseignement de tronc commun (INF3002) de niveau L3 pour la formation des managers (Télécom École de Management). Ce module regroupe une promotion de 200 étudiants, pour lesquels il y avait 10 intervenants (titulaires, doctorants et vacataires) à gérer.

Entre 2009 et 2013 à TSP, j'ai eu la **responsabilité** d'un module de spécialité informatique (CSC5004, VAP ASR, Télécom SudParis) de niveau M2. Les 24 étudiants y apprennent les bases de l'informatique ambiante. La responsabilité inclut la gestion des 5 enseignants qui y interviennent, ainsi que la recherche d'intervenants extérieurs et la mise en place de conférences assurées par des industriels.