

Intergiciels pour applications distribuées sur réseaux dynamiques

Yves Mahéo

▶ **To cite this version:**

Yves Mahéo. Intergiciels pour applications distribuées sur réseaux dynamiques. Réseaux et télécommunications [cs.NI]. Université de Bretagne Sud; Université Européenne de Bretagne, 2011. tel-00633253

HAL Id: tel-00633253
https://theses.hal.science/tel-00633253

Submitted on 18 Oct 2011

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

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



Mémoire en vue de l'obtention du diplôme d'habilitation à diriger des recherches en informatique

devant

L'UNIVERSITÉ DE BRETAGNE-SUD
sous le sceau de L'université Européenne de Bretagne
École doctorale SICMA

présenté par

Yves Mahéo

Intergiciels pour applications distribuées sur réseaux dynamiques

le 21 avril 2011
devant le jury composé de :

M. Serge CHAUMETTE
Professeur des universités, Université de Bordeaux I / *rapporteur*

Mme Isabelle DEMEURE
Professeur, Télécom ParisTech / *rapporteur*

Mme Silvia GIORDANO
Professeur, Scuola universitaria professionale delle Svizzera italiana / *rapporteur*

M. Flavio OQUENDO
Professeur des universités, Université de Bretagne-Sud / *examinateur*

M. Michel RIVEILL
Professeur des universités, Université de Nice Sophia Antipolis / *examinateur*

Remerciements

Je remercie tout d'abord très sincèrement les rapporteurs de cette habilitation, Isabelle Demeure, Serge Chaumette et Silvia Giordano, pour leur lecture manifestement attentive de ce mémoire, leur analyse de mon travail et leurs commentaires clairvoyants. Je remercie également Michel Riveill, qui a accepté d'être examinateur, et Flavio Oquendo, qui a endossé le rôle de président de jury.

Il est clair que cette habilitation n'aurait pas pu aboutir sans le travail de toute une équipe. Je remercie donc tous mes collègues enseignants-chercheurs de l'équipe Casa du Valoria, en premier lieu Frédéric Guidec et Nicolas Le Sommer, mais aussi Luc Courtrai et Pascale Launay. L'étroite collaboration entre les membres de l'équipe et, de façon plus générale, l'ambiance de travail qui y règne, m'ont permis de mener à bien les projets que je décris dans ce document. J'associe bien évidemment aux résultats présentés les doctorants que j'ai encadrés, Didier Hoareau et Roméo Said. J'en profite pour remercier Patrice Frison d'avoir accepté d'être le directeur de la thèse de Didier et Pierre-François Marteau, directeur du Valoria, pour m'avoir toujours soutenu dans mes activités d'encadrement. Je remercie également pour leur contribution les autres doctorants de l'équipe Casa : Salma Ben Sassi, Julien Haillot et Hervé Roussain, ainsi que Mario Dragone, ingénieur de recherche.

Je ne serais pas complet sans adresser des remerciements spécifiques à Frédéric Guidec et Sébastien Lefèvre pour avoir amicalement accepté la tâche de relecture de mon manuscript et de s'en être acquitté avec le plus grand sérieux.

Résumé

Ce document présente les travaux que j'ai menés au Valoria depuis 2002 dans le domaine des intergiciels supportant les applications distribuées orientées composants et les applications distribuées orientées services s'exécutant dans des réseaux dynamiques.

Les réseaux cibles des applications distribuées ont connu une évolution significative ces dernières années, faisant apparaître un dynamisme croissant. Une première caractéristique des réseaux dynamiques est la volatilité, qui implique que certaines machines du réseau peuvent être amenées à ne plus participer à l'application, de façon temporaire ou définitive. Une autre caractéristique est apparue avec l'avènement de l'informatique mobile : dans un contexte où les machines sont mobiles et communiquent par radio, la portée limitée des transmissions induit de fréquents changements de topologie du réseau. Les travaux décrits dans ce document concernent deux catégories de réseaux dynamiques. Dans un premier temps, nous nous sommes intéressés aux applications relevant du Grid Computing et plus particulièrement aux applications parallèles ciblant des grappes non dédiées, c'est-à-dire à des ensembles de stations de travail hétérogènes banalisées reliées par des réseaux d'interconnexion eux aussi banalisés, offrant donc des performances variables. Dans un deuxième temps, nous avons considéré des réseaux cibles de l'informatique ambiante. Nous avons en particulier étudié les réseaux mobiles ad hoc discontinus, c'est-à-dire des réseaux formés spontanément à partir de machines mobiles communiquant par radio directement entre elles, sans passer par une infrastructure fixe, et dont la topologie est telle qu'ils ne se présentent pas sous la forme d'une seule composante connexe mais plutôt d'un ensemble d'îlots de communication distincts.

Pour faciliter le développement et l'exploitation des applications distribuées sur réseaux dynamiques, il apparaît utile de s'appuyer sur des paradigmes de programmation de haut niveau tel que ceux mis en avant dans l'approche orientée composants et l'approche orientée services. Ces approches permettent notamment un découplage entre les entités de l'application, facilitant la gestion de la complexité du développement et du déploiement des applications dans un environnement dynamique. La plupart des technologies de composants et de services ont été conçues pour des réseaux stables et ne conviennent généralement pas aux applications sur réseaux dynamiques. Les travaux décrits dans ce document ont pour objectif de faciliter l'exploitation des composants et services dans un contexte dynamique. Nous nous sommes surtout focalisés sur le support à l'exécution des applications bâties à partir de composants et services, ce support prenant la forme d'un intergiciel, c'est-à-dire d'un ensemble de services logiciels construits au-dessus des systèmes d'exploitation et des protocoles de communication, et invoqués par les composants de l'application.

Nos contributions sont présentées à travers trois projets principaux : le projet Concerto, portant sur la définition d'un modèle de composants parallèles associé à un intergiciel pour des applications devant être déployées sur des grappes de stations de travail banalisées ; le projet Cubik, étendant le modèle de composants Fractal et proposant un support pour le déploiement et l'exécution de composants ubiquitaires pour réseaux dynamiques ; et le projet Sarah, s'attachant à la construction d'une plate-forme à services bâtie au dessus d'un protocole de communication adapté aux réseaux mobiles ad hoc discontinus.

Table des matières

1 Introduction **9**

1.1 Thématiques de recherche 9

1.2 Projets 11

1.3 Contexte des recherches 12

1.4 Organisation du document 13

2 Approche orientée composants pour le développement d'applications sur réseaux dynamiques **15**

2.1 Introduction 15

2.1.1 Notion de composant logiciel 15

2.1.2 Support d'exécution des composants 17

2.1.3 Contribution au développement d'intergiciels à composants pour réseaux dynamiques 18

2.2 Composants parallèles adaptables pour le Grid Computing 19

2.2.1 Motivation 19

2.2.2 Modèle de composant 21

2.2.3 Déploiement d'un composant 22

2.2.4 Mécanismes d'adaptation 23

2.2.4.1 Découverte et observation des ressources : l'intergiciel D-Raje 25

2.2.4.2 Modélisation des ressources 25

2.2.4.3 Gestion distribuée des ressources 26

2.2.4.4 Découverte de ressources 27

2.2.4.5 Observation 28

2.2.5 Administration 29

2.2.6 Application à un code de routage actif 29

2.2.7 Bilan 32

2.3 Composants ubiquitaires 34

2.3.1 Motivation 34
2.3.2 Composants hiérarchiques ubiquitaires 36
2.3.3 Distribution 38
2.3.4 Support des déconnexions 40
2.3.5 Déploiement 42
2.3.5.1 Déploiement de composants ubiquitaires, vue d'ensemble de l'approche Cubik 43
2.3.5.2 Spécification 45
2.3.5.3 Processus de déploiement 48
2.3.6 Mise en œuvre 51
2.3.7 Bilan 52

3 Approche orientée services pour le développement d'applications sur MANET discontinus 55

3.1 Introduction 55
3.2 Réseaux mobiles ad hoc discontinus 56
3.2.1 MANET connexes et MANET discontinus 56
3.2.2 Communication dans les MANET discontinus 60
3.3 Approche orientée services 63
3.3.1 Concepts principaux de l'approche orientée services 63
3.3.2 Plates-formes à services 65
3.4 Plate-forme à services pour MANET discontinus 68
3.4.1 Architecture générale 68
3.4.2 Communication 69
3.4.2.1 Entités du protocole 70
3.4.2.2 Description du protocole 71
3.4.2.3 Mise en œuvre 74
3.4.3 Découverte de services 76
3.4.3.1 Architecture de découverte 76
3.4.3.2 Descripteurs de services 77
3.4.3.3 Processus de découverte 77
3.4.4 Invocation de services 80
3.4.4.1 Principe 80
3.4.4.2 Réduction de la redondance des requêtes et réponses 81

3.4.4.3 Invocation basée destination 83
3.4.4.4 Invocation de service avec état 84
3.4.5 Évaluation 85
3.4.5.1 Conditions de simulation 85
3.4.5.2 Évaluation de la découverte 86
3.4.5.3 Évaluation de l'invocation 87
3.4.6 Bilan 89

4 Conclusion **95**

Bibliographie **101**

A Encadrements doctoraux **117**

B Projets de recherche **119**

Table des matières

1 Introduction

Ce mémoire d'habilitation résume mes travaux de recherche effectués depuis 2002 au Valoria, le laboratoire d'informatique de l'université de Bretagne Sud. On peut situer ces travaux dans le domaine général du développement d'applications distribuées, à l'intersection des systèmes distribués, du réseau et du génie logiciel. Les problématiques étudiées ne sont pas toujours centrales à un domaine. Il me paraît donc utile de préciser brièvement dans cette introduction les thématiques qui sont abordées dans le document, ainsi que les projets concrets qui ont constitué le cadre de mes travaux. Je donne également quelques points de repère sur mon parcours, qui, sans doute, sont de nature à éclairer mon approche des problèmes de recherche auxquels j'ai été confronté.

1.1 Thématiques de recherche

Réseaux dynamiques

Une application distribuée (ou application répartie) est une application composée d'un ensemble de processus possédant leur propre mémoire locale et communiquant par passage de messages. Les réseaux cibles des applications distribuées ont connu une évolution significative ces dernières années. Initialement, ils étaient essentiellement formés d'ensembles de machines fixes et toujours en opération, interconnectées par des liens stables et fiables. De tels réseaux étaient des réseaux locaux ou des réseaux de grande envergure, formant typiquement une partie d'Internet, et une grande partie des travaux – notamment algorithmiques – sur les systèmes distribués ont fait l'hypothèse simplificatrice de la stabilité des réseaux visés. Puis, pour aller au delà de la « simple » tolérance aux pannes (objectif qui reste dans bien des cas très difficile à atteindre), on a peu à peu cherché à prendre en compte plusieurs formes de dynamisme inhérentes à un fonctionnement normal du réseau.

Une première forme de dynamisme provient de la volatilité : certaines machines du réseau, ou les processus qu'elles hébergent, peuvent être amenés à ne plus participer à l'application, de façon temporaire ou définitive. C'est par exemple le cas de figure envisagé dans les réseaux dits pair à pair sur Internet, ou dans le Grid Computing.

Une autre forme de dynamisme est apparue avec l'avènement de l'informatique mobile. En effet, dans un contexte où les machines sont mobiles et communiquent par radio, la volatilité reste en vigueur mais le dynamisme est alors également induit par la portée limitée des transmissions. Au gré de leurs mouvements, les machines perdent des voisins et en acquièrent de nouveaux, induisant donc des changements de topologie dans le réseau.

Les travaux décrits dans ce document concernent deux catégories de réseaux que l'on peut qualifier de dynamiques. Dans un premier temps, je me suis intéressé aux applications relevant du Grid Computing et plus particulièrement aux applications parallèles ciblant des grappes non dédiées, c'est-à-dire des ensembles de stations de travail hétérogènes banalisées reliées par des réseaux d'interconnexion eux aussi banalisés, offrant donc des performances variables. La volatilité des machines est dans ce cas un problème incontournable qui s'ajoute à celui de l'hétérogénéité des ressources impliquées. Dans un deuxième temps, j'ai considéré des réseaux cibles de l'informatique ambiante, qui proposent une vision dans laquelle un grand nombre d'ordinateurs et autres dispositifs communicants omniprésents interagissent entre eux et avec les utilisateurs. Il s'agit de réseaux constitués de machines hétérogènes potentiellement mobiles. J'ai étudié en particulier les réseaux mobiles ad hoc discontinus, c'est-à-dire des réseaux formés spontanément à partir de machines mobiles communiquant par radio directement entre elles, sans passer par une infrastructure fixe, et dont la topologie est telle qu'ils ne se présentent pas sous la forme d'une seule composante connexe mais plutôt d'un ensemble d'îlots de communication distincts.

Approche orientée composants et services

Il n'est clairement pas aisé de développer, exploiter et maintenir une application distribuée, à fortiori si elle cible le type de réseaux que nous avons cité plus haut dans un contexte de Grid Computing ou d'informatique ambiante. Le caractère distribué de l'application apporte son lot de complications du fait du parallélisme et de la concurrence des processus impliqués, complications aggravées par la nécessaire adaptation au dynamisme et à l'hétérogénéité de l'environnement d'exécution. Des avancées relativement récentes en génie logiciel sont de nature à faciliter le travail des développeurs, en proposant des paradigmes de haut niveau tels que les composants et les services.

Nos travaux ont apporté des contributions à l'étude de l'approche orientée composants et l'approche orientée services pour le développement d'applications distribuées, dans le cas où les réseaux cibles sont des réseaux dynamiques. Dans l'approche orientée composants, on conçoit une application comme un assemblage d'entités fonctionnelles (les composants) déployables indépendamment et reliées entre elles. Chaque composant définit contractuellement, à travers des interfaces, les fonctions qu'il fournit et les fonctions qu'il requiert de la part d'autres composants. L'objectif global est de renforcer l'indépendance entre les entités mises en jeu pour faciliter notamment la réutilisation de code et l'adaptation au dynamisme de l'environnement d'exécution. L'approche orientée services reprend en partie les concepts de l'approche orientée composants mais met l'accent sur le découplage fort entre les entités de l'application (les services) à l'exécution.

Alors que dans l'approche orientée composants, la spécification de l'assemblage est souvent de nature déclarative et effectuée lors de la conception de l'application, les liaisons entres services se font tardivement, à l'exécution, après une phase de découverte durant laquelle un service client prend connaissance des services fournis qui lui sont utiles et qu'il pourra invoquer.

Les technologies de composants et de services sont nombreuses et certaines sont déjà employées à un niveau industriel. La plupart d'entre elles ont cependant été conçues pour des réseaux stables et ne conviennent généralement pas aux applications sur réseaux dynamiques.

L'un des enjeux du domaine est d'identifier les points clés où des adaptations sont nécessaires afin de prendre en compte les spécificités des réseaux dynamiques et de proposer et valider des solutions alternatives portant sur des modèles ou des supports d'exécution.

Intergiciels

L'approche orientée composants ou services couvre un grand nombre de thématiques de recherche (aspects formels, architecture, programmation, modèles économiques…). Je me suis surtout focalisé sur l'étude du support à l'exécution des applications bâties à partir de composants et services. Ce support prend la forme d'un intergiciel, c'est-à-dire d'un ensemble de services logiciels construits au-dessus des systèmes d'exploitation et des protocoles de communication, et invoqués par les composants de l'application. Ces services couvrent par exemple des aspects liés au déploiement, à l'assemblage de composants, à la découverte de services ou à la communication. L'enjeu est d'identifier les abstractions appropriées et d'organiser les protocoles utilisés, les algorithmes, les modules logiciels en des plates-formes intergicielles aptes à faciliter le développement des applications, la réutilisation de code, l'adaptation aux changements d'environnement d'exécution.

1.2 Projets

Les thématiques citées plus haut ont été abordées lors de recherches menées dans trois projets successifs dont la description fait l'objet du cœur de ce document.

- *Concerto : Composants parallèles adaptables pour le Grid Computing (2002-2003)*
  L'objectif du projet Concerto était de définir un modèle de composants et de construire un intergiciel associé pour des applications parallèles devant être déployées sur des grappes de stations de travail banalisées. Nous avons défini un modèle de composants simple intégrant la notion de parallélisme et, dans l'optique de fournir aux composants parallèles des capacités d'adaptation, nous avons doté l'intergiciel Concerto de moyens d'introspection de son environnement matériel et logiciel.
- *Cubik : Composants ubiquitaires sur réseau dynamiques (2003-2007)*
  Les travaux du projet Cubik ont visé eux des environnements d'informatique ambiante. Notre objectif était de supporter des applications ubiquitaires c'est-à-dire des applications dont les fonctions sont accessibles de l'ensemble du réseau, tout en ciblant des réseaux dynamiques. À travers la proposition d'un modèle de composants hiérarchique issu de Fractal, et en construisant un intergiciel supportant l'exécution des composants ubiquitaires, nous avons abordé les aspects relatifs à l'architecture des composants, au support des déconnexions et au déploiement distribué.

– *Sarah : Fourniture de services sur réseaux mobiles ad hoc discontinus (2006-2010)*
Nos travaux sur les réseaux mobiles ad hoc discontinus nous ont conduits à nous intéresser à l'approche orientée service. Le découplage fort entre fournisseurs et clients de services est en effet de nature à faciliter la prise en compte du dynamisme important imposée par ce type d'environnement d'exécution. L'absence de connectivité de bout en bout dans les réseaux mobiles ad hoc discontinus est une contrainte forte qui rend difficile la simple communication entre terminaux. Notre approche a donc été dans un premier temps d'étudier les aspects relatifs à la communication, en proposant un protocole de dissémination épidémique de documents basée sur le contenu, mis en œuvre dans l'intergiciel DoDWAN. Nous avons bâti la plate-forme intergicielle DiSWAN en complétant ce protocole par des fonctions de découverte et d'invocation de services adaptées aux contraintes des réseaux visés.

1.3 Contexte des recherches

C'est dans le cadre de mon stage de DEA de l'IFISC (Université de Rennes I) en 1990 que j'ai effectué mes premiers travaux de recherche. Ce stage s'est déroulé au Computer Laboratory de l'université de Newcastle Upon Tyne, sous la direction de Pete Lee et la supervision de Jean-Pierre Banâtre et portait sur un support pour l'exécution parallèle de programmes Gamma sur une machine parallèle à mémoire partagée [r3].

J'ai été accueilli en septembre 1991 dans l'équipe Pampa de l'Irisa pour commencer ma thèse sous la direction de Françoise André et Jean-Louis Pazat [r2]. Mes travaux s'inscrivaient dans la thématique de la parallélisation automatique de programmes séquentiels. L'approche étudiée était celle de la compilation dirigée par les données. Dans cette approche, on part d'un programme impératif séquentiel annoté par des directives de distribution des données (essentiellement des tableaux) et on produit par compilation un code pouvant s'exécuter sur un réseau de machines distribuées, typiquement un supercalculateur ou une grappe de stations de travail. Le langage sur lequel s'appuyaient les travaux de l'équipe était, pour des raisons pratiques, un langage dérivé de C, même si HPF (*High Performance Fortran*) s'est finalement imposé comme le langage source principal du domaine. Mon travail de thèse s'est focalisé sur la construction d'un support d'exécution associé au compilateur développé dans l'équipe, l'ensemble formant l'environnement Pandore. Il s'agissait de définir et d'implanter un ensemble d'opérations constituant les cibles de plusieurs schémas de compilation qui visaient d'une part des machines parallèles à messages et d'autre part des machines parallèles dotées d'un système de mémoire virtuelle partagée. Ma contribution a principalement porté sur la conception d'optimisations de l'accès aux données et des communications, optimisations fondées sur un système de pagination logique des tableaux manipulés par le programme [j5, i28, i26, i27, i24, i22, i21]. J'ai également effectué un travail sur l'évaluation des performances de l'approche, dans un premier temps par une série d'expérimentations sur plusieurs applications de calcul scientifique et dans un deuxième temps par la définition d'outils de mesure et de visualisation adaptés à la compilation par distribution de données [n7, i28, i18].

Pendant cette période, j'ai aussi contribué au développement de la POM (*Parallel Observable Machine*), une bibliothèque de communication générique pour machines à mémoire distribuée, dotée de mécanismes d'observation de l'exécution des programmes répartis [j4, i23]. Cette

bibliothèque a servi de support de base pour l'exécutif Pandore mais aussi pour d'autres prototypes développés dans l'équipe.

J'ai pu poursuivre mes travaux à l'Irisa dans le prolongement de ma thèse alors que j'occupais un poste d'ATER à l'INSA de Rennes en 1995-96, dans le cadre du développement de la bibliothèque Cidre [n6, n5, i19, i20]. Cette bibliothèque reprenait la mise en œuvre de la distribution de tableaux fondée sur la pagination logique de Pandore pour offrir une API de manipulation de tableaux utilisable dans des programmes explicitement répartis.

En 1997, j'ai rejoint le laboratoire d'informatique Valoria en étant nommé maître de conférences à l'Université de Bretagne-Sud à Vannes. Il s'en est suivi une période peu propice à la production de résultats de recherche, que l'on peut expliquer par une énergie essentiellement consacrée au montage de modules d'enseignements dans une toute jeune université, et à la mise en place d'axes de recherche dans un laboratoire tout aussi jeune (l'UBS et le Valoria ont été crées en 1995). J'ai pu toutefois mettre en place en 1999 une activité de recherche sur les grappes de stations de travail, en collaboration avec Frédéric Raimbault et Luc Courtrai. Notre objectif était de proposer un support pour la communication rapide d'objets dans des programmes distribués s'exécutant sur des grappes de stations de travail homogènes [i17, i16]. Nous avons repris certains concepts définis dans l'exécutif Pandore et dans la bibliothèque Cidre pour les appliquer non pas à des tableaux, mais à des objets Java sérialisables.

C'est à partir de 2002 que j'ai repris une activité de recherche plus soutenue alors que la structuration du laboratoire atteignait un palier. Le Valoria s'est alors structuré de manière à concentrer ses activités autour du thème général de l'informatique ubiquitaire et de l'intelligence ambiante, à travers la définition de trois axes thématiques : les systèmes logiciels interactifs multimédia et intelligents, l'architecture des systèmes logiciels, et les intergiciels pour les systèmes distribués mobiles et communicants. J'ai participé à la création et l'animation de ce dernier axe, qui est porté par l'équipe Casa. Les travaux décrits dans ce mémoire ont tous été menés dans l'équipe Casa, en collaboration avec mes collègues maîtres de conférences (Nicolas le Sommer, Frédéric Guidec, Luc Courtrai jusqu'à 2006, Pascale Launay depuis 2009) et ingénieur (Mario Dragone depuis 2009) et les doctorants que nous avons collectivement encadrés (Hervé Roussain, Didier Hoareau, Julien Haillot, Roméo Said, Salma Ben Sassi, Djamel Benferhat).

1.4 Organisation du document

La suite de ce mémoire comporte trois chapitres.

Le chapitre 2 regroupe la description de nos travaux ayant trait à l'approche orientée composants. Le paragraphe 2.1 constitue une introduction aux concepts de cette approche et résume notre contribution générale. Dans le paragraphe 2.2, on s'intéresse à l'utilisation des composants logiciels pour le Grid Computing, en particulier à la définition et au support de composants dits parallèles. L'essentiel de cette partie est consacré à la description des résultats obtenus au cours du projet Concerto. Le paragraphe 2.3 est intitulé « Composants ubiquitaires » et concerne l'utilisation de l'approche orientée composants pour des applications d'informatique ambiante. Il porte sur nos travaux effectués dans le cadre du projet Cubik sur la définition d'un modèle hiérarchique de composants ubiquitaires et la construction d'un intergiciel supportant l'exécution d'applications utilisant de tels composants dans des réseaux dynamiques. Les problématiques de déploiement sont notamment détaillées dans cette partie.

Le chapitre 3 est consacré à notre étude de l'approche orientée services pour le développement d'applications sur réseaux mobiles ad hoc discontinus. Le paragraphe 3.1 introduit rapidement les thématiques abordées. Le paragraphe 3.2 présente les caractéristiques des réseaux mobiles ad hoc discontinus et les problématiques de communication associées. Le paragraphe 3.3 constitue une brève introduction à l'approche orientée services et donne quelques repères sur la construction de plates-formes à services dans les réseaux dynamiques et en particulier dans les réseaux mobiles ad hoc. Enfin le paragraphe 3.4 détaille notre contribution en décrivant d'une part un protocole de communication adapté aux réseaux mobiles ad hoc discontinus et d'autre part la plate-forme DISWAN dédiée à la fourniture de services, et qui exploite ce protocole de communication.

Le chapitre 4 conclut ce mémoire en dressant un bilan général du travail mené et en évoquant quelques perspectives qu'il ouvre.

2

Approche orientée composants pour le développement d'applications sur réseaux dynamiques

2.1 Introduction

2.1.1 Notion de composant logiciel

La notion de composant logiciel, dont les prémices remontent maintenant à plusieurs décennies [106], est communément reconnue comme pouvant constituer le fondement du développement d'applications complexes, en particulier du développement d'applications distribuées. L'ingénierie logicielle orientée composants a connu un développement récent significatif avec l'essor de technologies telles que les EJB de Sun (*Entreprise Java Beans* [113]), les composants Corba de l'OMG (*Corba Component Model* [123]) ou .Net de Microsoft [111].

L'ambition du développement orienté composants est de pallier les lacunes de l'approche « tout objet » tout en proposant un niveau d'abstraction supérieur. La programmation orientée objet fonde la réutilisation essentiellement sur les notions d'héritage et de polymorphisme. Cependant, un couplage fort est maintenu entre les objets, induit par la présence au sein même du code d'un objet de références vers d'autres objets. Ceci rend difficile l'identification des dépendances entre les objets et va à l'encontre du principe de substitution. En outre, la partie métier de l'application, c'est-à-dire l'ensemble des fonctionnalités propres à cette application et les services transversaux aux applications (appelés aussi services non fonctionnels comme ceux relevant de la persistance ou de la sécurité) sont modélisés par des objets qui se révèlent fortement couplés, rendant difficile la maîtrise complète de l'application ainsi que son évolution, par manque de séparation des préoccupations.

Définition d'un composant logiciel

L'objectif visé par l'approche composant est de renforcer la réutilisation. À l'instar de ce qui peut se faire dans des domaines tels que l'électronique ou la construction de bâtiments, on veut pouvoir construire des applications par assemblage de composants, notamment de composants standard pris sur étagère, et faire évoluer ces applications par simple substitution de composants. La réduction des coûts de développement et l'augmentation de la fiabilité résultent de la réutilisation la plus large possible de composants développés indépendamment les uns des autres.

Pour répondre à cet objectif, un composant doit posséder les caractéristiques essentielles suivantes [139, 109] :

- Il intègre une description de son usage permettant aux autres composants de l'utiliser.
- Il est composable, c'est-à-dire qu'il peut être lié à d'autres composants.
- Il est conditionné de telle sorte qu'il puisse être déployé indépendamment des autres composants.

Ces caractéristiques se retrouvent dans la majorité des propositions académiques et industrielles faisant intervenir la notion de composant. Ces propositions concernent tout d'abord ce que l'on peut appeler des *modèles de composants*, c'est-à-dire des ensembles de définitions formalisant la notion de composant et les interactions entre composants [88]. Ces modèles sont souvent associés à des outils logiciels permettant l'exécution effective d'applications à base de composants, outils qui sont désignés par le terme d'*intergiciels à composants* (on parle aussi parfois d'*infrastructures à composants*). Le vocable *technologie de composants*, sans doute plus flou et à connotation industrielle, est aussi employé pour parler d'un intergiciel à composants supportant un modèle à composants plus ou moins explicite.

Architecture de composants

L'un des apports majeurs de l'approche à composants est de permettre de décrire une application comme un assemblage de composants. Il ne s'agit pas simplement de lister les composants impliqués, mais de définir une architecture de composants décrivant les entités qui devront être assemblées (les composants) et explicitant les interactions entre les différents composants. De nombreux travaux sur les modèles de composants, menés à l'origine dans la communauté du génie logiciel, se sont efforcés de formaliser les assemblages de composants à travers l'utilisation de langages appropriés appelés ADL *(Architecture Description Languages)* [107].

La majorité des ADL considère des graphes dans lesquels les nœuds représentent des unités de calcul (les composants) et les arêtes des connexions entre composants. Dans certains cas, la composition peut être hiérarchique, c'est-à-dire que les composants peuvent être eux-mêmes des assemblages de composants (par exemple dans Darwin [99] et Fractal [19]). On parle alors de modèles de composants *hiérarchiques* par opposition aux modèles de composants *à plat* (comme CCM ou les EJB).

La description de l'usage d'un composant passe par la définition de ses *interfaces*. Un composant possède des interfaces *fournies* décrivant les services qu'il implante. On définit également les interfaces *requises* par un composant pour exprimer les dépendances de ce composant vis-à-vis

d'autres composants. Une interface est constituée d'opérations (ou méthodes) éventuellement complétées par un ensemble de propriétés. La notion de port est parfois introduite, avec un rôle similaire à la notion d'interface ou comme abstraction de celle-ci. Les interfaces requises et fournies représentent le contrat d'assemblage d'un composant : lors de l'assemblage des composants, on met en correspondance les interfaces requises d'un composant avec des interfaces fournies d'autres composants. On établit alors des liaisons entre composants, représentées par des connecteurs pouvant prendre plusieurs formes, de la simple référence au composant-connecteur intégrant la gestion des modalités d'interactions relevant par exemple de la synchronisation lors des appels de méthodes ou de la définition du protocole de communication (cf. [22]). Par exemple dans l'ADL Wright [5], un connecteur prend la forme d'une entité typée réutilisable (on peut donc le voir comme un composant lui-même) qui décrit les interactions entre les composants qui lui sont connectés. La description spécifiée, qui met en jeu des rôles (les rôles attendus des composants) et une glue (qui combine les rôles), est exprimée en CSP [67].

2.1.2 Support d'exécution des composants

Intergiciel à composants

Il est possible de mettre l'accent sur l'utilisation des composants lors de la phase de conception d'une application, en utilisant un modèle de composants à travers un ADL. Cependant, un modèle de composants est le plus souvent associé à un support d'exécution qui permet de manipuler également à l'exécution les composants de l'application en tant que tels. Ce support prend la forme d'un intergiciel (ou *middleware*), c'est-à-dire d'une couche logicielle mettant en œuvre une abstraction des systèmes d'exploitation et des protocoles pour offrir un ensemble de services liés à l'utilisation des composants. Les applications sont développées typiquement dans un langage à objets et utilisent l'intergiciel via une API *(Application Programming Interface)* dédiée.

Les intergiciels à composants forment une des familles des intergiciels conçus pour la programmation des applications réparties, à côté des intergiciels à messages (ou *message-oriented middleware*), et des intergiciels à services (*service-oriented middleware*) [84, 46].

Les principaux services assurés par les intergiciels à composants sont les suivants.

- *Nommage et courtage*
  Le service de nommage sert à identifier un composant d'après son nom. Le courtage étend ce service en permettant une recherche de composant d'après des propriétés qui lui sont attachées.
- *Cycle de vie*
  La gestion du cycle de vie permet d'assurer la transition entre les différents états d'un composant. L'ensemble des états considérés varie d'un intergiciel à l'autre. On prendra par exemple en compte l'installation du code du composant, son instanciation, son activation (qui dénote sa capacité à recevoir des invocations) et sa passivation (qui, à l'inverse, le met dans un état inapte à recevoir des invocations). La persistance est une propriété non fonctionnelle associée au cycle de vie. Elle permet la conservation de l'état d'un composant sur un support de stockage pérenne afin de pouvoir reprendre son exécution ultérieurement, typiquement après un arrêt et redémarrage de l'intergiciel.

– *Assemblage*
  Qu'il soit ou non décrit préalablement via un ADL (dans une *configuration*), l'assemblage des composants de l'application doit être réalisé à l'exécution, au démarrage de l'application. Il s'agit de mettre en place les liaisons entre composants. Ceci est fait automatiquement à partir d'une configuration. L'intergiciel permet généralement de modifier en cours d'exécution l'assemblage via une API d'établissement et d'arrêt de liaison.
– *Communication*
  L'exploitation des liaisons entre composants se fait via des interactions synchrones (invocations de méthodes) ou asynchrones (publication / souscription d'événements). Les intergiciels à composants facilitent ces opérations en les rendant les plus transparentes possibles pour le programmeur (notamment en ce qui concerne l'appel de méthode), que ces interactions soient locales (entre composants hébergés sur une même machine) ou distantes (entre composants hébergés sur des machines distinctes). Les supports intergiciels permettant ces fonctions en programmation objet (RMI, Corba…) sont évidemment largement utilisés pour ce faire.

Des fonctions plus transversales qualifiées de services techniques ou propriétés non-fonctionnelles interviennent également dans la fourniture de ces services principaux. On supportera par exemple la gestion de la sécurité (authentification, chiffrement…) ou l'utilisation de transactions dans les interactions entre composants.

Conteneurs

De nombreux intergiciels à composants emploient la notion de *conteneur* à composants pour rassembler un certain nombre des services cités ci-dessus. Un conteneur encapsule les instances d'un ou plusieurs types de composants : il en assure le cycle de vie, et les interactions avec les autres composants passent par lui. Le conteneur est paramétrable, ce qui rend possible l'application de propriétés non fonctionnelles aux composants qu'il héberge. Par exemple, avec les EJB, on pourra mettre en place une politique de *pooling* pour la création des instances de composants, ou imposer un comportement transactionnel pour les invocations aux composants.

2.1.3 Contribution au développement d'intergiciels à composants pour réseaux dynamiques

Depuis 2002, l'équipe Casa s'est intéressée à l'approche orientée composants en se focalisant sur la production d'intergiciels à composants pour réseaux dynamiques. Je présente dans la suite de ce chapitre deux volets des travaux menés dans ce contexte.

Le premier volet, décrit dans le paragraphe 2.2, concerne les applications de Grid Computing. Il s'agit de permettre l'utilisation de composants dit « parallèles » déployés sur des grappes de stations de travail. Une des particularités de ces travaux est qu'ils prennent en compte l'hétérogénéité des grappes en intégrant dans un intergiciel à composants des mécanismes d'introspection sur les ressources accessibles aux composants, qu'il s'agisse de ressources matérielles ou logicielles.

Le deuxième volet, qui a suivi chronologiquement le premier, est détaillé dans le paragraphe 2.3. Il décrit des travaux qui visent la construction et l'exploitation d'applications à base de composants hiérarchiques dans un contexte d'informatique ubiquitaire. Un modèle de composants

complet inspiré de Fractal a été défini, ainsi qu'un intergiciel supportant l'exécution et le déploiement des composants sur des réseaux dynamiques.

Dans les deux cas, la vision de l'utilisation des composants logiciels comme constituants des applications distribuées est la même : chaque composant est potentiellement distribué sur plusieurs nœuds du réseau et l'assemblage de ces composants forme en conséquence une application bien évidemment distribuée. Cette vision se démarque de celle communément admise dans laquelle les composants sont dit distribués car ils sont déployés individuellement sur des machines différentes du réseau et interagissent ensuite via le réseau.

2.2 Composants parallèles adaptables pour le Grid Computing

2.2.1 Motivation

Grilles de calcul et grappes

Le Grid Computing a pour but d'exploiter les capacités de calcul fournies par un ensemble potentiellement très grand d'ordinateurs dispersés géographiquement et couvrant plusieurs zones administratives. L'objectif est d'être capable de profiter de ressources cumulées (essentiellement en termes de puissance de calcul et de stockage) pour une application donnée. Une grille de calcul apparaît comme un agrégat de réseaux autonomes mobilisables pour effectuer un calcul. Elle est notamment caractérisée par l'hétérogénéité et la dynamicité des ressources mises en jeu.

Les grappes de calcul forment très souvent des éléments de base dans ces grilles. Une grappe est constituée d'ordinateurs plus ou moins banalisés – et donc relativement peu coûteux – interconnectés par un réseau local. Les grappes censées concurrencer les supercalculateurs sont des grappes dédiées, homogènes et s'appuyant sur un réseau très performant. De telles grappes sont en effet depuis plusieurs années les « machines parallèles » les plus puissantes. Cependant il existe également des grappes de stations de travail constituant des plates-formes hétérogènes et dynamiques dans la mesure où elles font intervenir des machines non dédiées, qui peuvent être temporairement indisponibles, et dont le réseau d'interconnexion peut également offrir des performances de communication variables.

Le déploiement, l'exécution et la maintenance d'applications de Grid Computing restent à l'heure actuelle des tâches difficiles. De nombreux travaux de recherche ont été menés sur le développement d'intergiciels adaptés à ce contexte, et des projets majeurs (*e.g.* Globus [54] ou Unicore [142]) convergent vers une standardisation, à travers notamment l'Open Grid Forum [1]. Cependant les applications opérationnelles que ces intergiciels permettent de développer ne possèdent souvent que des architectures simples, par exemple fondées sur le paradigme maître-esclaves.

1. http://www.ogf.org

Approche à composants pour les grilles

Une des voies de recherche pour bâtir des applications de Grid Computing complexes est de s'appuyer sur l'approche à composants. Les intergiciels tels que CCM ou EJB permettent de mettre en place des applications fondées sur des *composants répartis*. Chaque instance de composant est un code s'exécutant sur une machine donnée et interagissant avec d'autres composants de même nature, locaux ou distants. Cette approche n'est pas forcément bien adaptée aux grilles de calcul et aux grappes si l'on veut pouvoir tirer parti d'un code non seulement réparti mais aussi *parallèle*. Une alternative est de s'appuyer sur des composants qui soient eux-mêmes parallèles.

Un *composant parallèle* est un composant logiciel (il possède les caractéristiques citées plus haut) mais dont le code métier est un code parallèle censé s'exécuter sur un ensemble de machines. Un exemple typique de composant parallèle est celui d'un composant englobant un code SPMD (*Single Program Multiple Data*) exploitant le parallélisme de données et s'exécutant sur une grappe.

Peu de modèles supportent directement de tels composants parallèles. Parmi les travaux prenant en considération cet aspect, on peut citer le *Common Component Architecture* (CCA) [4] qui définit un modèle de composants dédiés aux applications scientifiques parallèles. L'objectif principal de CCA est de permettre l'interopérabilité de codes scientifiques pré-existants. Dans cette architecture, l'accent est mis sur la définition d'un langage de définition d'interface scientifique (SIDL) indépendant de tout langage de programmation, et les spécifications d'un modèle de communication entre composants à base de ports. Le projet Padico [41] (mené dans le cadre de l'ACI GRID-RMI 2001) étant lui aussi dédié au couplage de codes scientifiques, il présente des similarités avec CCA. Il étend le modèle de composants Corba (CCM) dans lequel un composant est associé à un unique espace d'adressage, la communication entre composants consistant alors à transférer les données d'un espace d'adressage à un autre via un mécanisme de communication. Un composant parallèle est défini comme étant une collection de composants CCM séquentiels qui exécutent en parallèle tout ou partie de ses services (modèle d'exécution SPMD).

Les travaux décrits dans [12] permettent de définir des composants parallèles aussi bien selon le modèle SPMD que MIMD, en s'appuyant sur la bibliothèque Pro-Active [9]. Une implémentation du modèle de composants Fractal a été réalisée qui permet la définition de composants hiérarchiques pour les grilles de calculs [105]. Des mécanismes pour la communication asynchrone, la migration d'activités, le déploiement et la mise au point sont offerts. L'introspection sur les composants et le placement des activités est possible mais de façon hétérogène : l'introspection sur la structure des composants est implicite au modèle Fractal, Pro-Active fournit quant à lui des mécanismes permettant de gérer le placement des objets actifs.

Projet Concerto

Les différents travaux présentés ci-dessus s'efforcent tous, par des approches différentes, de définir un modèle de composants pour les grilles de calcul et de proposer un intergiciel supportant ce modèle. Nous avons pour notre part mené au cours de la période 2002-2003 des travaux sur les composants parallèles adaptables, dans le cadre du projet ACI GRID Concerto. Les paragraphes qui suivent décrivent les résultats obtenus durant ce projet. Notre objectif était de proposer un modèle de composant parallèle pouvant être déployé sur une grappe de machines, tout en développant les méthodes et outils permettant à ce type de composant de s'adapter aux

spécificités matérielles et logicielles de la grappe cible. L'accent a été en effet placé sur les capacités d'adaptabilité du composant plutôt que sur la définition d'un nouveau modèle de composant complet. En règle générale, les intergiciels proposés dans les autres travaux ne permettaient pas de récupérer de manière précise les informations relatives à l'environnement d'un composant (dont le système d'exploitation) et de ses sous-composants. Partant de l'hypothèse que de telles informations sont nécessaires à l'adaptation, nous sommes efforcés de fournir un cadre homogène en associant des fonctionnalités d'observation de ressources à la définition d'un modèle de composants approprié pour le développement d'applications distribuées.

Le projet CONCERTO a fait l'objet du développement d'une plate-forme intergicielle permettant de valider notre approche. L'architecture générale de l'ensemble déployé sur chacune des machines de la grappe est illustrée dans la figure 2.1. Deux éléments principaux ont été construits : un intergiciel mettant en œuvre le support d'exécution des composants parallèles et un intergiciel de gestion des ressources nommé D-RAJE *(Distributed Resource-Aware Java Environment)*.

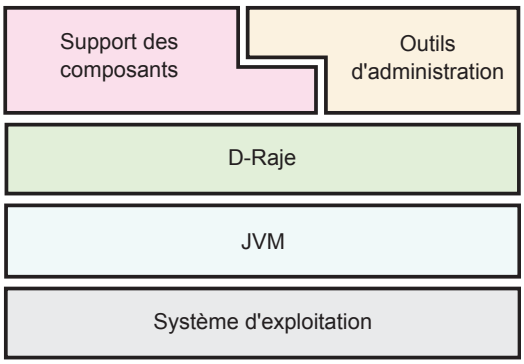


Figure 2.1 – Architecture de la plate-forme CONCERTO (sur une machine)

2.2.2 Modèle de composant

Un programmeur qui désire développer un composant parallèle dans CONCERTO doit concevoir son composant comme un ensemble de threads Java coopérants. Il doit en particulier définir la partie métier de son composant (ses nom, interface et implantation), la plate-forme lui offrant des facilités pour gérer les aspects non fonctionnels du composant.

Interfaces du composant

Le composant parallèle supporté par CONCERTO possède trois interfaces :

- *Interface métier*
  Aucune contrainte n'est mise sur le type de l'interface métier. Le programmeur choisit le type d'interface pour son composant, dans les possibilités offertes par Java. Il peut par exemple proposer une interface RMI pour la partie métier de son composant. Cette phase est indépendante de CONCERTO et demeure totalement à la charge du programmeur.
- *Interface cycle de vie*
  Cette interface proposée par CONCERTO permet de gérer les différentes étapes de la vie

du composant. Elle permet de déployer le composant sur les nœuds de la plate-forme en fonction de directives données par le programmeur ou l'administrateur ; d'activer le composant en lançant les threads qui le composent ; de stopper le composant, *i.e.* arrêter ses activités.

– *Interface ressource*
Le composant exhibe une interface ressource qui permet à l'utilisateur d'accéder aux informations relatives aux ressources offertes par la plate-forme Concerto et utilisée par les composants. La notion de ressource englobe les ressources système mais aussi des ressources dites « conceptuelles » telle que les composants, les fragments ou les threads. L'interface ressource peut donc être utilisée comme une interface d'introspection de l'architecture d'un composant. La mise en œuvre des ressources est réalisée par la couche D-Raje. Le composant, en tant que ressource D-Raje, doit implanter la méthode *observe()* qui retourne un rapport d'observation sur le composant. Par défaut, le rapport d'observation généré est un agrégat de l'ensemble des rapports d'observation des ressources qui le composent. Si le programmeur du composant souhaite contrôler ces informations, par exemple pour des raisons de sécurité, il peut restreindre les informations de son composant vues par les autres composants en définissant un type particulier de rapport d'observation.

Structure interne d'un composant

Lors de la construction d'un composant parallèle, le programmeur développe un ensemble de threads Java qui doivent coopérer pour implanter la partie métier du composant.

Les threads peuvent être réunis dans des entités de placement appelées fragments. Un fragment est un ensemble de threads appartenant au même composant, et le fragment avec ses threads seront forcément placés sur un même nœud de la plate-forme, et fonctionner au sein d'une même JVM. Les threads d'un même fragment peuvent partager des objets, communiquer et se synchroniser comme les threads de tout programme Java multi-threadé. Par contre, les threads de fragments différents ne peuvent interagir que via des mécanismes externes fournis par Java, tels que les sockets ou les RMI.

La figure 2.2 montre un ensemble de composants en cours d'exécution sur la plate-forme Concerto. La plate-forme regroupe trois nœuds et héberge quatre composants. Sur chaque nœud est lancé l'exécutif Concerto, dans une JVM. Un nœud peut contenir plusieurs fragments d'un même composant. Les threads d'un même fragment se partagent une mémoire d'objets et les fragments sont isolés les uns des autres.

2.2.3 Déploiement d'un composant

La phase de déploiement d'un composant consiste à placer les différents fragments sur les nœuds de la plate-forme. Concerto s'appuie sur un fichier de description du déploiement du composant fourni par le concepteur du composant. Ce fichier est écrit dans un dialecte XML. Il contient les informations suivantes :

– la structure du composant, c'est-à-dire la liste des fragments et pour chacun d'eux les noms des classes des threads qui le composent ;

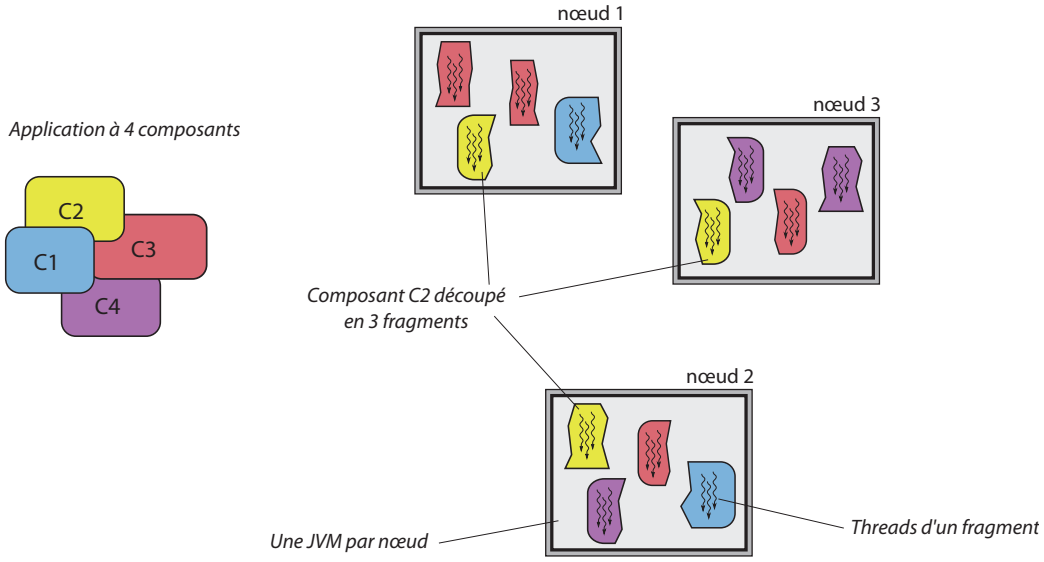


Figure 2.2 – Exemple de découpage de composants

- des directives de placement (par exemple une consigne de duplication d'un fragment sur tous les nœuds de la plate-forme, ou le placement d'un fragment sur une machine spécifique de cette plate-forme) ;
- des contraintes relatives au déploiement du composant. Elles concernent les pré-requis comme par exemple les services devant être installés pour pouvoir exécuter le composant (version spécifique de la JVM, présence d'un registre RMI ou d'un autre composant).

La figure 2.3 est un exemple de descripteur de déploiement d'un composant parallèle mettant en œuvre un algorithme génétique suivant le paradigme maître-esclave.

2.2.4 Mécanismes d'adaptation

L'environnement d'exécution des applications de Grid Computing est un environnement changeant. Ceci le différencie fortement des environnements classiques de déploiement des applications scientifiques, longtemps cantonnés aux supercalculateurs dont les caractéristiques étaient clairement définies et stables (parfois même au point que les applications étaient écrites en fonction de ces caractéristiques). Le partage de ressources entre applications, les opérations d'administration ou les pannes engendrent des fluctuations dans la disponibilité des moyens de calcul et de communication.

L'adoption d'une architecture à composants facilite l'adaptation des applications de Grid Computing. Elle apporte en effet de la souplesse lors du déploiement de l'application. Comme on l'a vu dans le cas du déploiement dans Concerto, une configuration spécialisée, qui tient compte des spécificités de la plate-forme cible, peut être choisie. Toutefois, pour tenir compte des fluctuations intervenant au cours de l'exécution, il convient de conférer aux composants des capacités d'adaptation dynamique. Ce type d'adaptation requiert des opérations que l'on peut classer en trois étapes :

- *L'observation*
  Il s'agit pour le composant d'obtenir dynamiquement des informations sur son environne-

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<component>
    <description>Genetic Algorithm Component - Function approximation</description>
        <name>
            <concerto-name>GenAlgoRes</concerto-name>
            <rmi-name>GeneticAlgorithmServer</rmi-name>
        </name>
    <requirements>
        <java kaffe="1.0.7" d-raje="1.0"/>
        <concerto-vers="1.0"/>
        <rmiregistry/>
    </requirements>
    <file>GenAlgo.jar</file>
    <fragment-list>
        <fragment name="master">
            <node>ANY</node>
            <thread-list>
                <runnable-class="ThMaster"/>
            </thread-list>
        </fragment>
        <fragment name="slave">
            <node>ALL</node>
            <thread-list>
                <runnable-class="ThSlave"/>
            </thread-list>
        </fragment>
    </fragment-list>
</component>
```

Figure 2.3 – Descripteur de déploiement du composant parallèle

ment d'exécution et d'évaluer l'état dans lequel se trouve l'application. On collectera par exemple des données sur le nombre de processus, la quantité de mémoire libre, la bande passante disponible entre deux nœuds.

– *La décision*
  Les éléments d'observation permettent de décider de la nouvelle conformation dans laquelle devrait se trouver l'application et des moyens d'y parvenir.

– *L'action d'adaptation*
  Il s'agit ensuite de réaliser un certains nombre d'opérations pour parvenir à un état adapté aux nouvelles conditions (création ou suppression de processus, changement d'algorithme de calcul ou de protocole de communication…)

La maîtrise du processus complet d'adaptation est une tâche extrêmement difficile dans le cas général et les travaux de recherche s'attachant à appliquer l'ensemble des étapes d'adaptation à des applications à composants restent peu nombreux [21, 3, 43, 16]. Parmi les problèmes à résoudre, on peut citer par exemple la coordination entre les différents composants d'une application distribuée pour garantir la cohérence de l'état dans lequel l'application se retrouve après adaptation ou bien la maîtrise des cycles d'hystérésis liés aux modifications des conditions d'exécution après les adaptations.

Les travaux du projet Concerto se sont focalisés sur la première étape du processus d'adaptation, l'observation du contexte d'exécution. Cette observation s'appuie sur l'intergiciel D-Raje.

2.2.4.1 Découverte et observation des ressources : l'intergiciel D-Raje

D-Raje est un intergiciel ouvert et extensible, développé dans l'équipe Casa, avec lequel un système distribué peut être modélisé à l'aide d'objets Java qui réifient les différentes ressources offertes par ce système. D-Raje constitue un élément du projet Concerto mais est mis en œuvre de telle sorte qu'il soit utilisable dans un autre contexte.

De façon générale, nous qualifions de ressource toute entité matérielle ou logicielle qu'un composant logiciel est susceptible d'utiliser pendant son exécution. L'approche utilisée dans D-Raje pour la modélisation des ressources est assez similaire à celle adoptée dans le *Common Information Model* [42]. Cependant D-Raje va au-delà de la simple modélisation. En plus de permettre que les ressources soient modélisées et gérées à travers des objets Java, D-Raje fournit des mécanismes permettant aux composants applicatifs :

- de découvrir l'existence de ressources spécifiques (ou de types de ressources spécifiques) dans leur environnement ;
- de rechercher des ressources spécifiques (ou des types de ressources spécifiques) dans leur environnement ;
- d'obtenir des informations sur l'état des ces ressources, suivant plusieurs modalités (observation directe ou notification sur événement).

D-Raje consiste en un cadre de conception pour la modélisation et l'observation des ressources. Contrairement aux approches visant la sécurité des applications (*e.g.* [37, 49, 7, 30, 8]), D-Raje ne se limite pas à un ensemble fixe de ressources relevant du système mais cherche à faciliter l'intégration et l'observation de tout type de ressource. Le contexte distribué est pris en compte dans D-Raje dans la mesure où les mécanismes cités ci-dessus sont implémentés de telle sorte que chaque ressource puisse être identifiée et observée de façon homogène indépendamment de sa localisation.

Les objectifs de D-Raje sont différents de ceux de la plupart des outils de modélisation et de monitoring de ressources proposés dans les projets de Grid Computing [85]. Que ceux-ci reposent sur l'utilisation d'annuaires (*e.g.* Globus [38], Condor [132]) ou suivent une approche objet (par exemple Legion [31], Javalin [117]), les ressources considérées sont essentiellement des ressources de gros grain tels que des nœuds de calcul ou des unités de stockage. L'objectif de ces environnements est surtout d'être capable de collecter des informations sur des ressources disséminées afin d'ordonnancer l'exécution d'un certain nombre de calculs intensifs. L'approche retenue dans D-Raje se veut plus générique, similaire à celle suivie dans d'autres travaux plus récents, appliqués aux composants logiciels, comme Lewys [25] ou Cosmos [34]. Ce dernier s'appuie d'ailleurs en partie sur un sous-ensemble de D-Raje.

2.2.4.2 Modélisation des ressources

L'architecture de D-Raje est organisée autour d'une hiérarchie de classes dans laquelle une classe Java est définie pour chaque type de ressource. La figure 2.4 montre quelques classes déjà définies. Elles réifient des ressources « système » (CPU, mémoire, swap, interfaces réseaux) qui caractérisent principalement la plate-forme matérielle sous-jacente et des ressources, qualifiées de « conceptuelles », qui concernent plutôt l'environnement applicatif vu au niveau de la programmation objet Java (sockets, threads, fichiers, répertoires…). Dans le cadre du projet

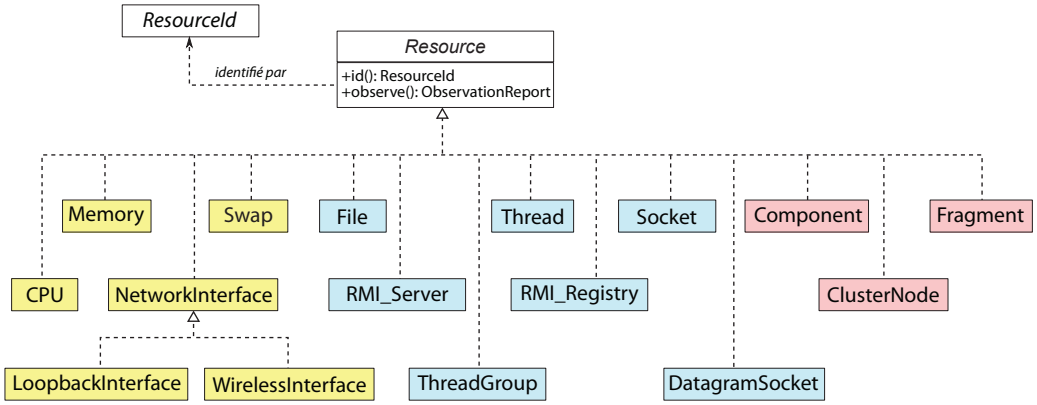


Figure 2.4 – Modélisation objet des ressources

Concerto, D-Raje a été étendu pour inclure dans l'ensemble des ressources des entités spécifiques au modèle de composant parallèle (composant, fragment). Ces nouvelles ressources conceptuelles permettent l'introspection de l'architecture des composants parallèles afin d'en faciliter l'adaptation en cours d'exécution.

Tout type dont la classe est censée réifier un type de ressource dans D-Raje doit implémenter l'interface *Resource*. Comme l'illustre la figure 2.4, cette interface sert de racine à la hiérarchie des classes ressources mais spécifie aussi qu'un objet ressource doit avoir un identificateur et qu'il doit être capable de fournir un rapport d'observation à la demande.

2.2.4.3 Gestion distribuée des ressources

D-Raje gère chaque objet ressource créé sur un nœud de la plate-forme distribuée. Cette gestion est assurée par un élément central de D-Raje appelé le gestionnaire de ressource (voir figure 2.5). Cet élément joue le rôle d'un annuaire distribué sur la plate-forme et est accessible à l'utilisateur à travers une instance de la classe *ResourceManager*. Chaque objet de type *ResourceManager* instancié localement coopère avec ses homologues dans la plate-forme distribuée afin d'offrir une vue unique du gestionnaire de ressources. L'objectif est de fournir un accès homogène à l'information sur les ressources quel que soit leur emplacement.

Dans l'implantation réalisée, une seule JVM est présente sur chacun des hôtes de la plate-forme, et un seul objet *ResourceManager* est instancié dans cette JVM. Le gestionnaire de ressources offre plusieurs services distribués :

- il enregistre toute création ou destruction d'objet ressource ;
- il identifie et repère chaque objet ressource ;
- il permet à l'utilisateur d'appliquer divers critères de sélection afin d'obtenir les identités d'un sous-ensemble particulier des ressources enregistrées ;
- il permet à l'utilisateur d'obtenir des rapports d'observation sur une ou plusieurs ressources.

Les objets ressources peuvent être crées manuellement dans un programme d'application. Alternativement l'instanciation d'un ensemble de ressources peut être gérée automatiquement

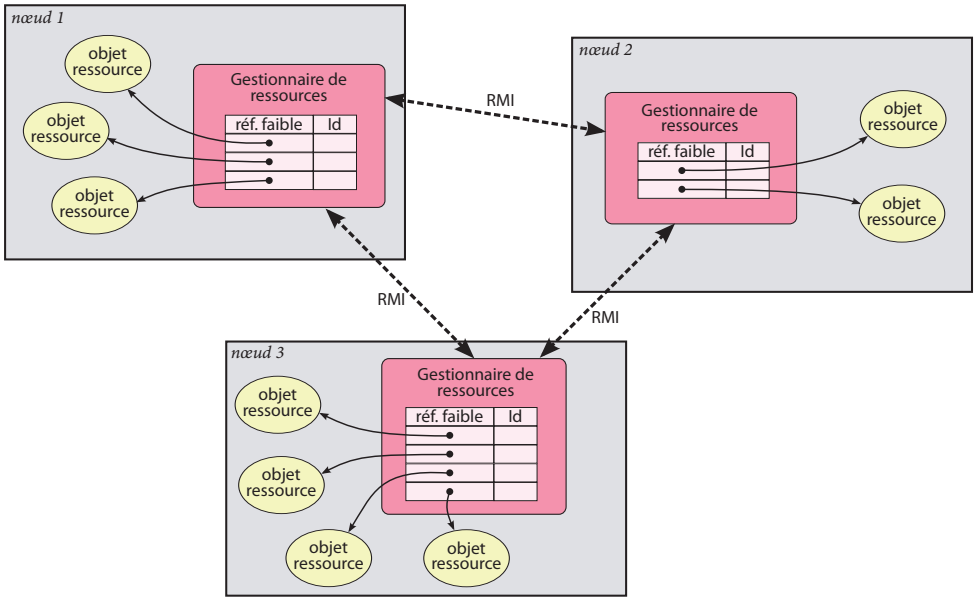


Figure 2.5 – Architecture du gestionnaire de ressources

ou semi-automatiquement. Cela est utile par exemple pour des ressources système sporadiques comme les interfaces réseau sans fil : des moniteurs spécialisés peuvent être facilement créés pour gérer des objets ressources de type *WirelessInterfaceResource*, créés et détruits automatiquement au gré des fluctuations des interfaces système correspondantes.

D-Raje offre des mécanismes pour aider à identifier et retrouver les ressources à l'exécution. Ils reposent sur un système de nommage qui attribue à chaque ressource un identifiant unique. Dans cette optique, D-Raje propose essentiellement un cadre de conception, offrant donc la possibilité de plusieurs mises en œuvre. Dans l'implantation actuelle de D-Raje, un système de nommage à plat est utilisé, mais des systèmes de nommage et d'enregistrement complexes (par exemple hiérarchiques) peuvent être déployés.

2.2.4.4 Découverte de ressources

Le nombre de ressources dans un système distribué peut être très grand. Afin de faciliter la découverte des ressources adéquates pour une application donnée, nous avons défini une collection de motifs qui permettent de rechercher des ressources de façon sélective, avec des critères portant sur la localisation des ressources ou sur les caractéristiques même des ressources.

En ce qui concerne la recherche des ressources selon leur localisation, un ensemble de classes de motifs permettent de se focaliser sur une fraction de la plate-forme distribuée, de limiter une opération du gestionnaire de ressources aux nœuds voisins ou de faire en sorte que toute la plate-forme soit considérée. La portion de code suivante montre comment on peut demander au gestionnaire de ressources de rechercher des ressources suivant différents motifs. Dans cet exemple, plusieurs motifs sont employés pour indiquer que la recherche s'applique (1) seulement aux ressources locales ; (2) aux ressources situées sur un hôte distant dont l'identité est passée en paramètre ; (3) à toutes les ressources, quel que soit leur emplacement.

```
ResourceManager manager = ResourceManager.getManager();
Set<ResourceId> localIds = manager.getResourceIds(new LocalSearch());  // (1)
Set<ResourceId> remoteIds =
     manager.getResourceIds(new LocalSearch(remoteNodeId));  // (2)
Set<ResourceId> allIds = manager.getResourceIds(new GlobalSearch());  // (3)
```

La sélectivité sur les caractéristiques des ressources s'appuie aussi sur une collection de classes de motifs correspondant généralement chacune à une classe de ressources. Ces motifs permettent des recherches très fines, éventuellement personnalisables via une redéfinition de la méthode *isMatchedBy()* fournie par chaque classe de motif. L'exemple suivant montre la création d'un motif qui permet de sélectionner les objets ressources qui modélisent des sockets et qui satisfont les critères suivants : l'adresse IP de l'hôte distant doit appartenir au réseau 195.83.160/24 et le port distant doit être dans l'intervalle 0 à 1023. D'autre part, l'adresse IP locale et le port sur lequel la socket est liée peuvent prendre ici n'importe quelles valeurs.

```
[...]
ResourcePattern socketPattern =
         new SocketPattern(InetAddress.AnyAddress, "195.83.160/24",
                           PortRange.AnyPort, new PortRange(0, 1023));
Set<ResourceId> socketIds = manager.getResourceIds(socketPattern);
```

2.2.4.5 Observation

Observation synchrone

Un objet ressource (implantant l'interface *Resource*) définit des méthodes spécifiques qui permettent la consultation de l'état courant de la ressource qu'il modélise. Par exemple, un objet de type *CPUResource* disposera d'une méthode permettant de connaître la fréquence du CPU. Par ailleurs, afin de faciliter le monitoring d'ensembles hétérogènes de ressources en utilisant un schéma unique et générique, la méthode *observe()* déclarée dans l'interface *Resource* fournit un point d'accès générique pour demander un rapport d'observation à une ressource de n'importe quel type. Un rapport d'observation capture et conserve l'état atteint par une ressource à un instant donné. Une hiérarchie de classes Java a été développée afin de définir différentes sortes de rapports d'observation, et des outils inclus dans D-Raje permettent la génération, la collecte et la gestion de tels rapports. Le code ci-dessous, qui poursuit les extraits précédents, montre comment on peut obtenir un rapport d'observation sur une ressource préalablement identifiée (ici l'élément d'indice *i* d'un tableau de ressources-sockets).

```
[...]
ObservationReport report = manager.getObservationReport(socketIds[i]);
```

Observation asynchrone

En plus de fournir un mode d'observation directe qui s'exprime par l'envoi de requêtes au gestionnaire de ressources pour obtenir des données spécifiques ou des rapports d'observation

sur ces ressources, D-Raje permet de déléguer l'observation à des moniteurs paramétrables. En effet, en exploitant l'implantation d'un modèle d'événements simple mais flexible, on peut demander à un objet moniteur d'observer périodiquement certaines ressources pour être notifié de la survenue d'événements spécifiques concernant l'état de ces ressources.

Les modalités d'observation suivent un modèle souscription-publication. Dans un premier temps, le programme d'application effectue une étape de souscription en décrivant les événements qui l'intéressent. Ceci se fait par la définition de profils d'observation. Le programmeur peut ensuite instancier un objet moniteur en lui passant un ou plusieurs profils d'observation. Le moniteur aura en charge de faire les observations nécessaires. Il détectera l'occurrence des événements choisis et publiera ces événements, aboutissant à la notification des objets cibles spécifiés dans les profils d'observation. Le placement du moniteur peut être choisi suivant les besoins. Par exemple on peut choisir de placer le moniteur sur le nœud où sont générées les données d'observation, auquel cas la notification d'événement se fera à travers le réseau. On peut aussi placer le moniteur localement, et dans ce cas chaque collecte de rapport d'observation se fera à distance.

2.2.5 Administration

L'environnement Concerto a été doté d'un ensemble d'outils fonctionnant en ligne de commande ou disposant d'une interface graphique permettant l'administration de la plate-forme, le déploiement de composants et leur observation. L'outil graphique *ConcertoTool*, se présente sous la forme d'un tableau de bord (voir les figures 2.6 et 2.7). Il montre en un seul écran les nœuds faisant partie de la plate-forme, les composants déployés avec les ressources D-Raje liées à Concerto (composants, fragments, threads) avec pour chacune d'elles le nœud où elle est hébergée. Les composants peuvent être déployés par l'intermédiaire d'une boite de dialogue de gestion de fichiers. L'outil comprend une fenêtre de type terminal affichant les traces des actions effectuées sur l'exécutif. Un menu reprend l'ensemble des commandes ; il permet par exemple d'ajouter des nœuds ou d'activer des composants.

2.2.6 Application à un code de routage actif

À la fin de l'année 2003, Concerto a été utilisé pour encapsuler dans un composant parallèle adaptable un code de routage actif développé au sein de l'équipe RESO du LIP (ENS Lyon). Il s'agissait d'un élément de l'environnement Tamanoir, développé dans le cadre des recherches de cette équipe sur les réseaux actifs orientés haute performance. L'objectif de ce travail était de démontrer l'intérêt de Concerto pour la définition et le déploiement d'un code parallèle sous forme de composant logiciel, et surtout d'évaluer les facilités offertes au programmeur pour rendre son composant adaptable.

Routage actif dans Tamanoir

L'environnement Tamanoir se propose de répondre aux problèmes d'hétérogénéité et de déploiement dynamique de services posés par l'utilisation de réseaux actifs [93, 53]. Il fournit aux utilisateurs la possibilité de déployer et de maintenir des routeurs actifs appelés TAN (*Tamanoir*

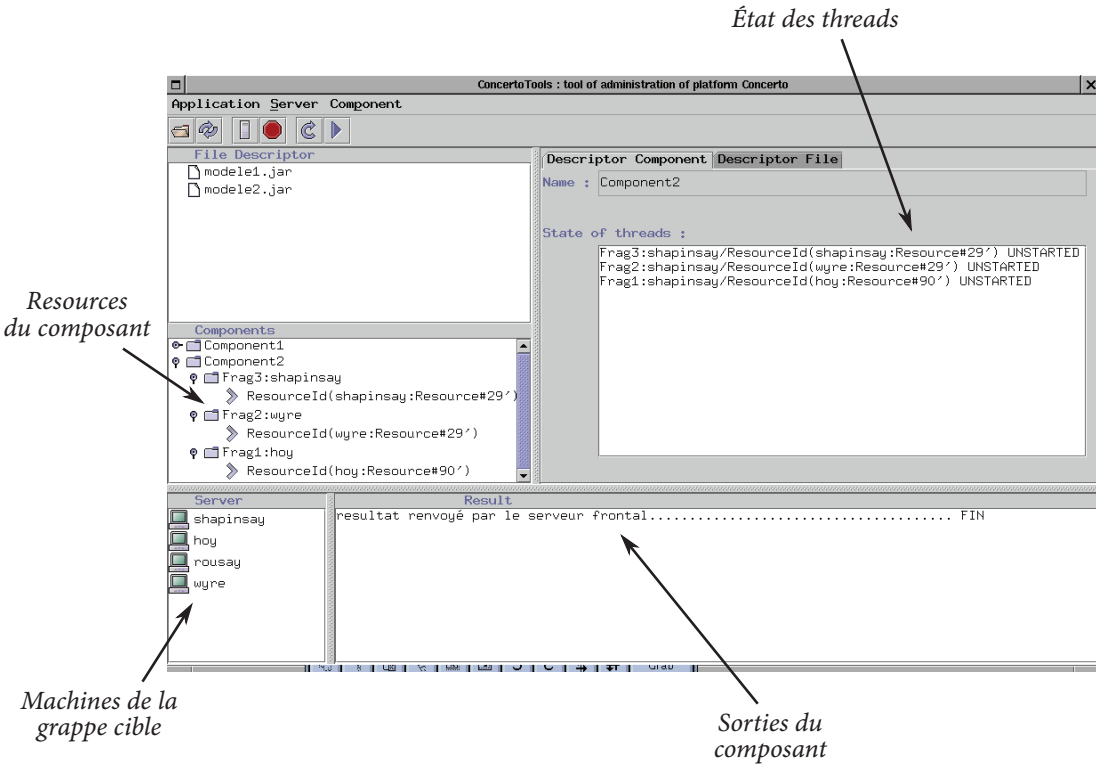


Figure 2.6 – Interface graphique ConcertoTool.

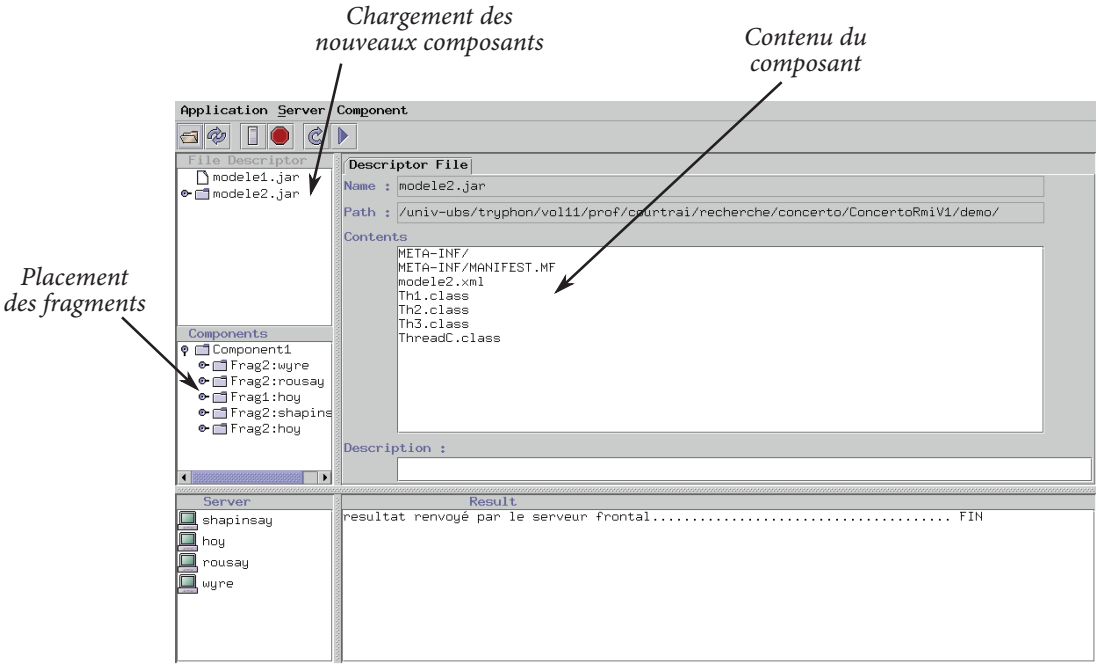


Figure 2.7 – Déploiement d'un composant Concerto avec ConcertoTool

Active Node), distribués sur un réseau à grande échelle. Les routeurs actifs, qui permettent d'appliquer dynamiquement des services aux flux qui les traversent, sont disposés à la périphérie du cœur du réseau (*core network*) qui doit rester passif pour assurer de très hautes performances.

L'architecture de Tamanoir est basée sur celle d'une grappe. Un nœud, appelé frontal, apparaît comme un routeur au sein du cœur du réseau. C'est ce nœud qui accepte des connexions multiples et est chargé d'orienter les flux de données vers les services adéquats. La nature du service à appliquer est spécifiée dans l'entête du flux. Le frontal gère également les besoins du TAN (téléchargement ou mise à jour de services, routage entre nœuds actifs…) et applique des services réseaux légers tels que le *forwarding* simple ou le marquage qui nécessitent peu de traitement sur les paquets de données. La puissance de calcul fournie par les nœuds de la grappe qui sont reliés au frontal permet d'appliquer les services lourds (compression à la volée, chiffrement, filtrage…). La grappe dispose de liens à très haut débit pour assurer une distribution efficace des flux.

L'un des enjeux de l'approche Tamanoir est d'offrir la possibilité d'utiliser des services ouverts (de nouveaux services doivent pouvoir être facilement mis en œuvre), portables, chargeables et déployables dynamiquement. Pour certains d'entre eux, on peut avoir des contraintes particulières qui imposent de tirer parti de ressources locales au routeur actif (mémoire, disque, processeur de traitement dédié…).

Le développement de l'environnement Tamanoir a été fait essentiellement en Java afin de disposer d'un langage portable et courant, permettant aux utilisateurs de réseaux actifs de définir et développer un service pour leurs propres besoins. Concrètement, un service est représenté par une classe Java implantant une interface générique *Service*.

Composant Concerto Tamanoir

L'absence de contrainte sur la définition de la partie métier d'un composant Concerto a facilité la reprise du code Tamanoir pour obtenir un composant parallèle. L'architecture du composant Tamanoir est de type maître-esclave. Le code du composant se découpe en deux types de fragments : un premier fragment contient le code du frontal (fragment *frontend*) et un second type de fragment (fragment *backend*) décrit le code Tamanoir chargé d'appliquer les services.

Le fragment frontal est évidemment destiné à être déployé en un seul exemplaire, sur le nœud frontal de la grappe. Il comporte initialement trois threads, chargés respectivement de traiter les flux TCP, de traiter les datagrammes UDP et d'implémenter la politique de sélection d'un *backend*.

Le fragment *backend* est censé être dupliqué sur les nœuds banalisés de la grappe. Il ne comporte qu'un thread initialement. Celui-ci identifie dans un premier temps le *frontend* afin d'être prêt à recevoir les flux et datagrammes de sa part. À l'ouverture d'un flux TCP ou à la réception d'un datagramme UDP, il charge via le *frontend* le code de la classe du service demandé si cela est nécessaire.

Le déploiement ne pose pas de problème particulier. Un descripteur de déploiement similaire à celui de la figure 2.3 a été utilisé pour indiquer le placement des différents composants sur la grappe.

Adaptation et administration

L'intérêt de l'utilisation de Concerto pour développer le routeur actif distribué Tamanoir réside surtout dans la possibilité d'exploiter les mécanismes d'obtention d'information sur les ressources. Tout d'abord, il s'agit d'obtenir des informations provenant essentiellement du système afin d'effectuer un équilibrage de la charge affectée aux backends, et d'obtenir ainsi des performances suffisantes dans l'exécution des services. Il faut noter que, dans le cadre du projet Concerto, notre objectif n'était pas de déterminer une politique idéale. La détermination de celle-ci doit se faire par raffinements successifs à partir d'heuristiques de départ, ce qui nécessite des expérimentations multiples dans des contextes réalistes. En revanche, l'utilisation de Concerto (et donc de D-Raje) a permis de faciliter la mise en place de nouvelles politiques et donc accélérer la phase de raffinements successifs.

La manipulation de ressources conceptuelles de Concerto permet de faciliter le développement de mécanismes d'administration et de supervision d'un composant parallèle Tamanoir. En effet, plusieurs concepts de Tamanoir peuvent être considérés comme des ressources et donc faire l'objet d'une modélisation au sein de D-Raje. Le cadre de conception D-Raje a par exemple été appliqué aux notions de frontend et backend qui ont donc été assimilés à des ressources et par conséquent réifiés pour permettre leur manipulation explicite.

L'intégration des services Tamanoir dans D-Raje a été également étudiée. Une hiérarchie de ressources a été définie afin de modéliser les types de services Tamanoir à appliquer aux flux. Ces ressources sont créées lors du chargement de service sur les nœuds aptes à rendre le service. Une autre hiérarchie de classes modélise les instances des types de services. Ces ressources permettent de connaître à tout moment quels sont les services appliqués sur les backends et quelles en sont les caractéristiques (origine et destination du flux, cumul de consommation mémoire ou CPU…).

Une fois que les services sont réifiés sous la forme d'objets ressources, on peut exploiter l'infrastructure de D-Raje pour obtenir des informations sur les services, à l'intérieur de la grappe (par exemple sur le frontend) mais aussi à l'extérieur, en se plaçant comme client du composant TAN, via son interface ressource. On a ainsi développé un outil en ligne de commande permettant de connaître l'état du composant TAN s'exécutant sur une grappe, ceci étant un premier pas vers un outil plus complet d'administration et de mise au point du routeur.

2.2.7 Bilan

Nos travaux sur les composants parallèles adaptables ont été menés dans le cadre du projet Concerto, projet financé pour la période 2002–2003 par le Ministère de la Recherche dans le cadre du programme ACI GRID. Ce projet avait été sélectionné dans la catégorie « Jeune équipe », ce qui impliquait qu'il soit couvert par la seule équipe Casa, et excluait les partenaires pressentis lors de la réponse à l'appel d'offres (Irisa, Onera). Le projet Concerto, que j'ai dirigé, a impliqué plusieurs membres de l'équipe dont notamment Nicolas Le Sommer (alors doctorant encadré par Frédéric Guidec) dont le travail de thèse à contribué à définir un cadre de conception pour la contractualisation des ressources qui incluait les fonctionnalités d'observation de ressources à la base de D-Raje.

Le travail effectué durant le projet a donné lieu aux publications suivantes :

- Yves Mahéo, Frédéric Guidec, Luc Courtrai. Middleware Support for the Deployment of Resource-Aware Parallel Java Components on Heterogeneous Distributed Platforms. *In Proceedings of the 30th Euromicro Conference – Component-Based Software Engineering Track*, Rennes, France, Septembre 2004, pages 144–151, IEEE CS [**i12**].
- Yves Mahéo, Frédéric Guidec, Luc Courtrai. Towards Resource-Aware Parallel Components. *In Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA'04)*, Las Vegas, NV, USA, juin 2004, pages 1006–1012, CSREA Press [**i13**].
- Yves Mahéo, Frédéric Guidec, Luc Courtrai. A Java Middleware Platform for Resource-Aware Distributed Applications. *In Proceedings of the 2nd International Symposium on Parallel and Distributed Computing (ISPDC'03)*, Ljubljana, Slovénie, octobre 2003, pages 96–103, IEEE CS [**i14**].
- Luc Courtrai, Frédéric Guidec, Nicolas Le Sommer, Yves Mahéo. Resource Management for Parallel Adaptive Components. *In Proceedings of the Workshop on Java for Parallel and Distributed Computing, International Parallel and Distributed Processing Symposium (IPDPS'03*), Nice, France, avril 2003, pages 134–141, IEEE CS [**i15**].
- Luc Courtrai, Frédéric Guidec, Yves Mahéo. Concerto : gestion de ressources pour composants parallèles adaptables. In *Chapitre 4 des actes de l'École GRID'2002*, pages 41–53, Aussois, France, décembre 2002 [**n3**].
- Luc Courtrai, Frédéric Guidec, Yves Mahéo. Gestion de ressources pour composants parallèles adaptables. *In Actes des Journées Composants, Systèmes à composants adaptables et extensibles*, Grenoble, France, octobre 2002 [**n2**].

L'objectif du projet Concerto était de contribuer à l'étude de l'application de la technologie à composants logiciels au Grid Computing, en proposant un modèle de composant parallèle adaptable. Dans cette optique, nous avons défini un modèle simple de construction des applications à base de composants et un intergiciel permettant à ces composants de percevoir l'environnement distribué dans lequel ils s'exécutent. L'intérêt principal de l'intergiciel Concerto réside en effet dans la possibilité pour les composants de percevoir leur environnement d'exécution sous la forme d'un ensemble de ressources, qu'il s'agisse de leur environnement système ou des entités logicielles formant l'architecture de l'application.

Le modèle de composants proposé reste toutefois extrêmement simple dans la mesure où la notion d'assemblage de composants n'y est pas incluse. L'absence de prise en compte de l'assemblage se retrouve également à l'exécution puisqu'il n'existe pas de support intergiciel pour la communication entre composants, le programmeur devant recourir à des outils de communication entre objets distants tels que les RMI.

Comme nous le verrons dans le chapitre suivant, nous avons poursuivi nos travaux sur les composants distribués, dans la continuité du projet Concerto. En conservant l'idée de distribuer un composant sur plusieurs machines cibles, notre nouvel objectif était notamment de mieux prendre en compte les problématiques liées à l'architecture de l'application, sans pour autant proposer un modèle de composants complètement nouveau. À l'occasion du démarrage de ces travaux, effectués essentiellement dans le cadre de la thèse de Didier Hoareau, nous avons ciblé une classe de réseaux quelque peu différente en passant des environnements de Grid Computing aux environnements d'informatique ubiquitaire.

2.3 Composants ubiquitaires

2.3.1 Motivation

Parallèlement à l'émergence des grilles de calcul évoquées dans la première partie de ce chapitre, on a vu se développer ces dernières années des environnements dit d'informatique ubiquitaire. L'informatique ubiquitaire (ou informatique pervasive, ou encore informatique ambiante) vise à développer et exploiter des logiciels dans des environnements formés d'ordinateurs omniprésents (allant de la station de travail au *smartphone* en passant par les *PDA* et les *netbooks*) et pouvant inclure des éléments comme des capteurs ou des étiquettes électroniques. Tous ces équipements sont capables de communiquer entre eux, en permanence ou de façon sporadique, souvent à travers des liaisons sans fil. Les termes pervasif, ambiant et ubiquitaire expriment le fait que nous pouvons être baignés en permanence dans une myriade de tels ordinateurs communicants, souvent sans que nous en ayons conscience [149].

La recherche en informatique ubiquitaire n'est pas sans lien avec celle sur les grilles de calcul, au point d'ailleurs que plusieurs conférences couvrent conjointement les deux thèmes [2]. En effet les grilles de calcul et les environnements d'informatique ubiquitaire partagent des caractéristiques fondamentales telles que l'hétérogénéité des ressources, la volatilité des équipements, la fluctuation des capacités de communication… et le fait qu'ils constituent tous des cibles pour déployer des applications distribuées. La différence tient surtout à la mobilité des équipements, qui n'intervient quasiment pas dans les grilles de calcul, mais dont l'importance peut être grande dans l'informatique ubiquitaire. En général, on attend également une autonomie accrue de la part des applications ubiquitaires.

La diversité des environnements d'informatique ubiquitaire est extrêmement grande. À travers les travaux que nous décrivons dans la suite de ce chapitre, nous nous sommes intéressés à une catégorie d'environnements dans laquelle le caractère d'enfouissement des équipements n'était pas pris en compte (nous avons par exemple ignoré les contraintes énergétiques) et pour laquelle la mobilité des équipements était relativement faible (nous reviendrons sur ce dernier critère au chapitre 3), ce qui la rapproche d'une certaine façon des grilles de calcul. Nous utilisons le terme de réseaux dynamiques pour qualifier ces environnements. Il s'agit de réseaux d'ordinateurs caractérisés par les aspects suivants :

- *L'hétérogénéité*
  Les ordinateurs mis en jeu sont de nature variée : stations de travail puissantes, ordinateurs portables, assistants numériques personnels, smartphones… De même les technologies de réseaux sur lesquelles s'appuient les communications entre ces ordinateurs peuvent couvrir un large spectre : connexions filaires à très haut débit, liaisons Wi-Fi ou Bluetooth…
- *La sporadicité des possibilités de communication*
  Un premier facteur influençant la capacité d'un ordinateur à communiquer avec les autres est la volatilité. Les équipements portables ont des batteries qui limitent leur autonomie énergétique. Ils sont donc fréquemment éteints ou mis en veille, ce qui a un impact significatif sur les opportunités de communication.
  Par ailleurs la portée des transmissions radio est limitée. Un ordinateur actif n'est donc pas forcément en mesure de communiquer avec tous les autres ordinateurs du réseau à tout instant.

2. La cinquième conférence internationale « Grid and Pervasive Computing » a eu lieu à Taiwan en 2010.

Un exemple de réseau dynamique est présenté dans la figure 2.8. Ce réseau est constitué des ordinateurs auxquels accèdent régulièrement deux collègues de bureau, chez eux ou au travail. Il s'agit de plusieurs PC de bureau ou stations de travail, des portables, un mini-PC, et des PDA. La liaison à Internet à leurs domiciles est intermittente et une partie de l'utilisation de leurs portables sur leur lieu de travail se fait à travers un réseau Wi-Fi indépendant. On a donc là un réseau dynamique fragmenté en îlots de communication. Au sein d'un îlot, les équipements actifs sont susceptibles de communiquer, mais aucune communication n'est possible entre îlots distincts.

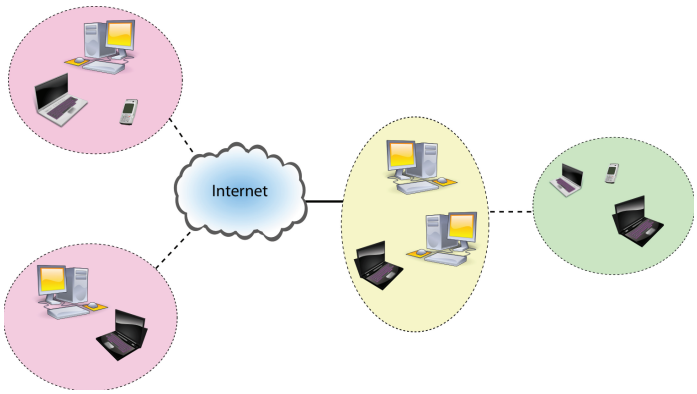


Figure 2.8 – Exemple de réseau dynamique

On peut comparer ce type de réseau à ceux que l'on peut voir qualifiés de dynamiques dans la littérature.

- Les environnements de Grid Computing sont constitués de machines relativement puissantes reliées par des réseaux rapides. Il sont dynamiques dans le sens où ils sont sujets à la volatilité ou à l'impossibilité d'accéder à une partie du réseau (par exemple pour des raisons administratives). Cependant on considère généralement qu'une application est déployée sur une partie du réseau pour laquelle il n'y aura pas de problème majeur de disponibilité ou de connectivité. Une étape de réservation de ressources est typiquement mise en œuvre pour se placer dans une telle situation.
- Les réseaux pair à pair sont généralement constitués d'ordinateurs banalisés reliés à Internet, et également sujets à la volatilité, du point de vue de la connexion effective de la machine. On considère cependant cette volatilité de façon binaire : une machine est soit dans le réseau (elle est active et peut communiquer avec toutes les autres), soit hors du réseau (elle est temporairement éteinte ou déconnectée).
- Dans les réseaux mobiles ad hoc (*Mobile Ad hoc NETworks* ou MANET), la mobilité et la portée radio limitée font qu'une machine change fréquemment de voisins. Cependant, on considère que la densité des machines est suffisante pour qu'un algorithme de routage permette à chaque machine de communiquer avec toutes les autres à tout instant.

Dans ces trois cas, la notion d'îlot de communication n'est pas prise en compte : une application est censée s'exécuter sur un réseau connexe. Au cours de nos travaux, nous nous sommes attachés au contraire à être capables de déployer et exécuter des applications distribuées sur des réseaux dynamiques pour lesquelles la fragmentation en îlots est commune.

Notre objectif était de proposer un support pour des applications dites ubiquitaires, c'est-a-dire des applications accessibles depuis n'importe quelle machine du réseau, supportant donc le caractère dynamique du réseau, y compris sa fragmentation en îlots. La conception et la réalisation d'un tel support ont fait l'objet du travail de thèse de Didier Hoareau [68], dans la continuité du projet CONCERTO décrit au paragraphe 2.2. Il s'agissait de proposer un modèle de composant plus complet, baptisé CUBIK, couvrant les aspects relatifs à l'architecture des applications et de compléter ce modèle par un support intergiciel pour l'exécution distribuée et le déploiement des applications.

2.3.2 Composants hiérarchiques ubiquitaires

Le modèle de composants CUBIK est un modèle de composants hiérarchique, dans lequel un composant peut être lui-même un assemblage de composants (on parle alors de composant *composite*). La récursion s'arrête avec les composants *primitifs* qui forment des unités encapsulant du code. Une application est alors généralement constituée d'un seul composant composite. Les modèles hiérarchiques permettent de travailler à différents niveaux d'abstraction. De la même manière qu'avec une approche fonctionnelle, le développement peut se faire par raffinements successifs en partant des composants composites de plus haut niveau jusqu'à arriver aux composants primitifs.

Nos travaux ne portaient pas sur les aspects de génie logiciel mais plutôt sur le développement de support intergiciel pour la distribution ; nous n'avions donc pas l'ambition de définir un modèle hiérarchique totalement nouveau. Il existe en effet plusieurs modèles de composants hiérarchiques qui pouvaient servir de base à nos travaux comme par exemple le langage de description d'architecture Darwin [101] associé à l'exécutif Regis [100] ou le modèle de composants Koala [144]. Nous avons construit notre modèle de composants en reprenant les concepts spécifiés par le modèle de composants hiérarchique Fractal [19]. Il faut noter que notre objectif n'était pas de rester compatible avec Fractal en proposant des extensions du modèle pour rendre les composants Fractal ubiquitaires. Nous avons plutôt repris certains aspects du modèle Fractal qui nous étaient utiles tout en nous autorisant à délaisser plusieurs caractéristiques principales de Fractal comme par exemple le partage de composants.

Modèle de composants Fractal

Le modèle de composants Fractal a été défini par France Telecom R&D et l'INRIA. Il se présente sous la forme d'une spécification (publiée pour la première fois en 2002) [20] et d'implantations dans différents langages de programmation (C, C++, Java…). Ce modèle est fondé sur les principes suivants :

- *Modélisation hiérarchique des composants*
  Les composants sont soit des composants composites, englobant d'autres composants, soit des composants primitifs.
- *Capacités d'introspection*
  L'objectif est de doter les composants de moyens de découvrir l'architecture dans laquelle ils se trouvent.

– *Capacités de configuration*
  L'objectif est de faciliter le déploiement d'une architecture de composants et de modifier cette architecture dynamiquement.

Un composant Fractal est une unité d'exécution possédant une ou plusieurs interfaces constituant chacune un point d'accès au composant et implantant un type d'interface (ensemble d'opérations supportées). Il existe deux catégories d'interfaces : les interfaces serveurs, qui correspondent aux services fournis par le composant, et les interfaces clients qui correspondent aux services requis par le composant.

Un composant primitif est une unité d'exécution (équivalente à un objet) présentant une ou plusieurs interfaces fonctionnelles serveur ou client. Un composant composite est constitué d'un contenu, d'un ensemble fini de sous-composants, et d'une membrane. À l'instar d'un composant primitif, la membrane possède des interfaces fonctionnelles. Elle inclut également des interfaces de contrôle permettant l'introspection et la configuration dynamique du composant. La membrane joue le rôle d'un conteneur. Elle est composée de contrôleurs ayant chacun un rôle particulier (fournir une liste ordonnée des sous-composants, gérer le cycle de vie du composant…).

Les interfaces d'une membrane sont soit externes, soit internes. Les interfaces externes sont accessibles de l'extérieur du composant, alors que les interfaces internes sont accessibles par les sous-composants du composant.

Le modèle Fractal fournit deux mécanismes permettant de définir l'architecture d'une application : l'imbrication (à l'aide des composants composites) et la liaison. La liaison permet aux composants Fractal de communiquer. Fractal définit deux types de liaisons : primitive et composite. Les liaisons primitives sont établies entre une interface client et une interface serveur de deux composants résidant dans le même espace d'adressage. Par exemple, une liaison primitive dans le langage Java est implantée à l'aide d'une référence. Les liaisons composites sont des chemins de communication arbitrairement complexes entre deux interfaces de composants. Les liaisons composites sont constituées d'un ensemble de composants de liaison (talons et squelettes par exemple) reliés par des liaisons primitives.

Une autre caractéristique originale du modèle de composants Fractal est qu'il permet la définition de composants partagés, qui appartiennent à plusieurs composites englobants. Ceci permet notamment de modéliser des ressources partagées, tout en conservant une forme d'encapsulation des composants. En effet, il n'est pas nécessaire à un composant de bas niveau d'exporter une interface au niveau du composite qui l'encapsule pour accéder à une interface d'un composant partagé.

Exemple d'application

Pour illustrer brièvement la modélisation d'une application sous la forme de composants Fractal, nous considérons ici une application de photographie spécialisée dans la création de diaporamas dont l'architecture est donnée dans la figure 2.9. Cette application est constituée d'un composant racine composite, appelé PhotoApp, incluant un composant générique dédié à la recherche de documents (DocumentSearch). Ce composant est aussi un composant composite (obtenu par exemple sur étagère) ; il englobe les composants primitifs DocumentFinder et DocumentBuffer. DocumentFinder offre une interface de recherche sur des documents de tout type,

permettant de trouver des documents en fonction de leur date de création, de méta-informations associées au document, etc., et une interface pour sélectionner les documents correspondants depuis un ensemble donné de documents. Les documents sélectionnés sont passés au composant DocumentBuffer qui les stocke. En plus d'une interface utilisable pour ajouter au stock de nouveaux documents, DocumentBuffer fournit une interface pour trier et extraire des documents. Cette interface serveur et celle de DocumentFinder sont accessibles en tant qu'interfaces serveurs du composant englobant DocumentSearch. Enfin, le composant DocumentBuffer est lié à un composant PhotoRepository prenant en charge la gestion d'une bibliothèque de photos et à un composant DiapoMaker qui permet d'assembler les photos sélectionnées pour fabriquer un diaporama paramétrable.

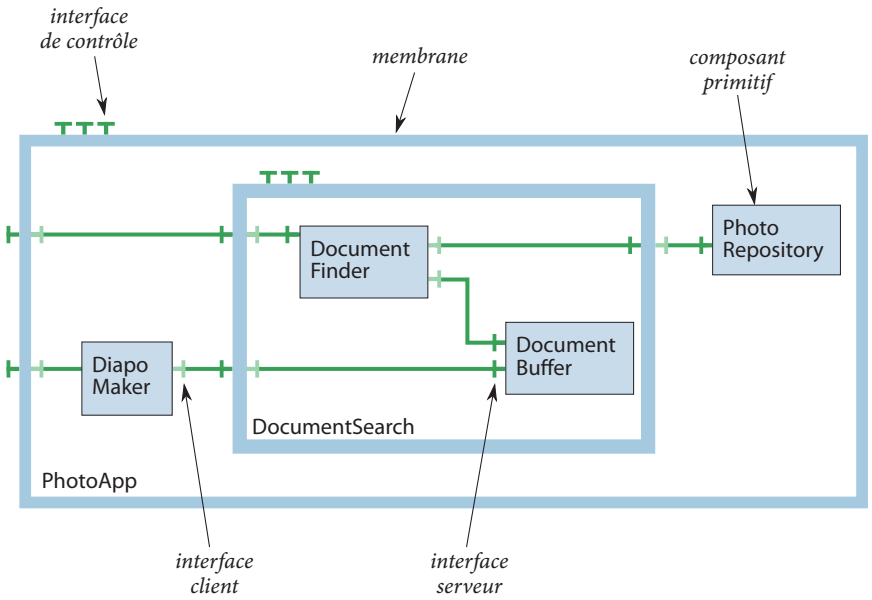


Figure 2.9 – Architecture Fractal de l'application de photo

2.3.3 Distribution

La raison d'être du modèle de composants Cubik est de rendre les fonctionnalités d'un composant hiérarchique disponibles depuis l'ensemble des machines d'un réseau dynamique. Cette accessibilité permet de qualifier le composant d'ubiquitaire. Nous avons pour cela caractérisé le mode de distribution d'un composant sur un ensemble de machines.

Dans notre approche, la définition d'un composant est couplée à son placement et cette relation est traitée différemment suivant qu'il s'agit d'un composant primitif ou d'un composant composite. On adopte les règles suivantes :

- À un composant primitif correspond une seule instance du code qui s'exécute sur une seule machine (on l'appellera la *machine cible*).
- À un composant composite correspondent plusieurs instances qui sont exécutées sur un ensemble de machines (on l'appellera l'*ensemble cible* du composite).

Une instance de composant composite est en fait une instance de sa membrane. Cubik reprend en effet la caractéristique des membranes de Fractal qui sont réifiées à l'exécution. Une membrane

dupliquée sur un ensemble de machines contient la liste des machines ainsi que l'état de la configuration des sous-composants directs du composite.

Composants ubiquitaires

Un composant C est dit ubiquitaire sur un ensemble cible M s'il existe sur toutes les machines de M une instance de C. L'ensemble M est choisi arbitrairement avec la contrainte qu'il doit être un sous-ensemble de l'ensemble cible du composant composite englobant. Si pour un composant, l'ensemble cible n'est pas précisé, c'est celui du composant englobant qui est considéré.

Lorsque l'on parle de la présence d'une instance de composant sur une machine, il s'agit d'une instance d'un type de composant. Cette instance peut éventuellement prendre la forme d'un mandataire du composant primitif ou de la membrane d'un composite.

Les ensembles cibles peuvent être définis par extension comme cela est fait dans l'exemple donné plus bas, où on liste les machines qui les composent. Nous évoquerons dans la partie du chapitre traitant du déploiement la possibilité de définir les ensembles cibles par compréhension, à l'aide de contraintes.

Qu'il soit primitif ou composite, tout composant ubiquitaire (hormis le composant racine) sur un ensemble cible M appartient à un composant composite englobant. Ceci impose la présence éventuelle de composants mandataires sur certaines machines de l'ensemble cible M mais aussi sur des machines de l'ensemble cible du composant englobant. Nous détaillons ici les deux cas de figure pouvant se présenter :

- *Cas d'un composant primitif ubiquitaire*
  Soit p un composant primitif ubiquitaire sur M, et contenu dans un composant englobant CE ubiquitaire sur ME. Par définition, on a $M \subseteq ME$. Une instance du composant p est présente sur chacune des machines de M. Sur une seule machine de ME, cette instance résulte de l'instanciation du code de p. Sur les autres machines de M, elle résulte de l'instanciation d'un composant mandataire possédant les mêmes interfaces clientes que p et capable de relayer les appels sur ces interfaces clientes vers l'instance unique distante du code de p.
- *Cas d'un composant composite ubiquitaire*
  Le schéma de distribution est similaire pour un composite. Soit C un composant composite ubiquitaire sur M, et contenu dans un composant englobant CE ubiquitaire sur ME, avec $M \subseteq ME$. Une instance de la membrane de C est présente sur toutes les machines. Cette instance est une instance d'une membrane complète sur chacune des machines de l'ensemble M. Pour les machines de ME qui ne sont pas dans M, il s'agit d'une instance d'un composant mandataire de membrane. Ce mandataire possède les mêmes interfaces qu'une membrane complète mais ne maintient pas d'information sur la configuration des sous-composants. Il se contente de maintenir une référence vers une des instances de membrane complètes de C. Ces dernières sont toutes identiques, le mandataire est donc lié à un exemplaire quelconque de ces membranes. Ainsi, plusieurs choix sont possibles pour définir au niveau du composant mandataire la machine hébergeant la membrane qu'il représente. Plusieurs stratégies peuvent guider ce choix (par exemple choisir la machine vers laquelle la bande passante est la plus large, choisir une machine accessible).

Exemple

La figure 2.10 illustre la distribution du composant de photo décrit à la figure 2.9. L'ensemble cible du composant racine PhotoApp est l'ensemble $\{m_1, m_2, m_3, m_4, m_5\}$. Un sous-ensemble $\{m_1, m_2, m_3\}$ de ces machines est dédié à l'exécution du composant DocumentSearch, m_4 et m_5 étant exclus pour des raisons de licence par exemple. De plus, des contraintes sur les ressources font que le placement des composants primitifs est le suivant : DocumentFinder sur m_1, DocumentBuffer sur m_2, PhotoRepository sur m_4 et DiapoMaker sur m_5.

Lorsqu'un composant n'est pas présent sur une machine, il est représenté par un mandataire figuré en pointillé. L'identité de la machine vers laquelle le mandataire maintient une référence est précisée. C'est le cas pour les composants primitifs : chacun de leurs mandataires est lié à une instance unique du code. C'est aussi le cas pour le composite DocumentSearch : sur les machines m_4 et m_5, ce composant est représenté par un mandataire lié, arbitrairement, à l'instance de DocumentSearch présente sur m_1.

2.3.4 Support des déconnexions

Le cycle de vie d'un composant comporte classiquement une étape dans laquelle le composant est qualifié d'actif c'est-à-dire apte à recevoir des appels sur ses interfaces fournies. Pour atteindre cet état, le composant doit pouvoir avoir lui-même accès aux composants dont il dépend. Les liaisons doivent avoir été établies entre toutes les interfaces requises du composant et les interfaces fournies des composants dont il dépend, et ces composants doivent eux-mêmes être actifs. Dans un contexte distribué, l'établissement des liaisons entre interfaces sous-entend que des liaisons distantes supposées suffisamment stables soient réalisées entre les machines hébergeant les composants impliqués. Or dans un environnement dynamique, les déconnexions réseau ne sont pas exceptionnelles et leurs occurrences risquent fort de rendre impossibles certaines liaisons entre interfaces de composants sur de longues périodes et du même coup empêcher l'activation du composant.

Ceci est d'autant plus pénalisant que la composition est hiérarchique. Dans un modèle comme Fractal, toute indisponibilité d'un composant primitif empêche le composant englobant de passer à l'état actif. Le composant englobant de niveau supérieur ne peut donc pas lui-même devenir actif et ainsi de suite jusqu'au composant racine. Au final, une application distribuée est entièrement inutilisable si un seul des composants primitifs est placé sur une machine qui devient inaccessible.

Pour éviter cette situation, nous avons défini un mode dégradé possible pour les composants hiérarchiques ubiquitaires. Dans ce mode dégradé, un composant peut rester actif même si tous les composants dont il dépend ne sont pas actifs. Pour cela nous appliquons le concept d'activité au niveau d'une interface individuelle.

L'idée sous-jacente à la définition d'une interface active est qu'une interface fournie d'un composant doit rester active si les invocations de méthode sur cette interface vont pouvoir aboutir, indépendamment de la possibilité d'utilisation d'une autre interface fournie de ce composant.

L'état d'une interface fournie d'un composant *C* dépend de l'état des interfaces fournies des composants requis par *C*, c'est-à-dire des composants liés aux interfaces requises de *C*. Il faut donc statuer sur l'existence d'une liaison entre les interfaces requises par *C* et fournies par d'autres,

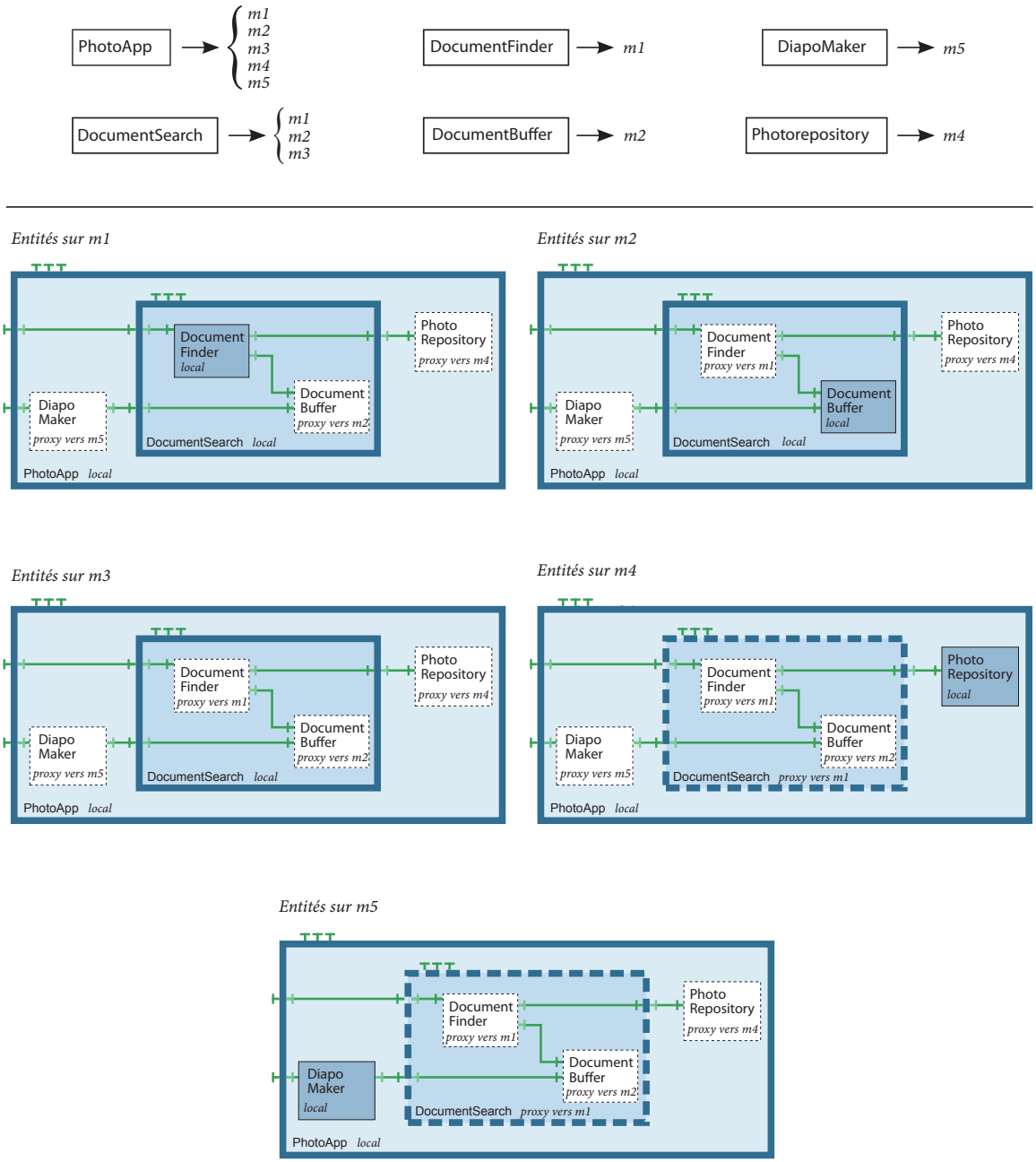


Figure 2.10 – Placement des composants et entités maintenues sur les machines m_1 à m_5

et sur la dépendance existant au niveau du composant C entre ses interfaces fournies et ses interfaces requises. Pour un composant composite, on peut établir précisément ces dépendances en analysant l'architecture interne du composant. Pour un composant primitif, vu comme une boîte noire, toute interface fournie est considérée comme étant dépendante de l'ensemble des interfaces requises.

On adopte les définitions suivantes :

- une interface requise est active si et seulement si elle est liée à une interface fournie active.
- une interface fournie d'un composant primitif est active si et seulement si toutes les interfaces requises de ce composant sont actives ;
- une interface (externe) fournie d'un composant composite est active si et seulement si l'interface requise (interne) correspondante est active.

Ceci permet d'établir pour une application les graphes de dépendances entre interfaces dans lesquels les sommets sont les interfaces et les arcs sont établis entre deux interfaces I_1 et I_2 liées par la relation « I_1 est active si et seulement si I_2 est active ». La figure 2.11 montre que pour le composant racine de notre application de photo, deux graphes disjoints sont obtenus. Le composant PhotoApp peut donc fonctionner en mode dégradé.

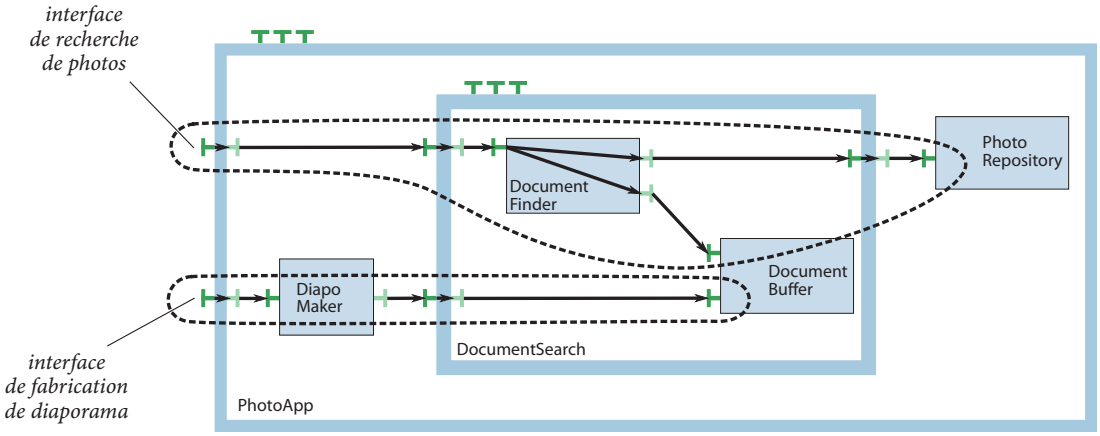


Figure 2.11 – Graphes de dépendances entre interfaces pour l'application de photo

Dans le cadre de la distribution décrite dans la figure 2.10, et si l'on considère par exemple l'utilisation du composant PhotoApp sur la machine m_1, dans le cas d'une rupture de connexion entre m_5 et m_1, il est possible de laisser active l'interface de recherche de photo (en haut sur la figure 2.11) même si l'interface de fabrication de diaporama doit être rendue inactive du fait de l'inaccessibilité de m_5.

2.3.5 Déploiement

Le cycle de vie d'un logiciel comporte plusieurs étapes couvrant les actions d'analyse, de conception, de production, de test, de déploiement et de maintenance. Le déploiement est en lui-même suffisamment complexe pour justifier des travaux de recherche spécifiques. Certains de ces travaux tels que [23] et [62] et plus récemment [96] ont permis d'identifier les différentes phases constituant le processus de déploiement d'une application, processus faisant intervenir les principales activités suivantes :

– *Mise à disposition*
  Il s'agit là de décrire l'application, d'assembler ses différents composants, de configurer ses paramètres et de préparer le ou les paquetages logiciels aptes à contenir le code et l'ensemble des informations associées. Les paquetages ainsi formés sont ensuite rendus disponibles, par exemple en les déposant dans un entrepôt.
– *Installation*
  Cette phase comprend tout d'abord le transfert de chacun des paquetages logiciels sur la machine sur laquelle il doit être exploité. Le code est ensuite extrait du paquetage et l'environnement d'exécution est configuré conformément aux directives extraites elles-aussi du paquetage.
– *Lancement*
  Chaque composant de l'application doit être instancié après une éventuelle configuration dynamique. Le cas échéant, ces instances de composant sont connectées entre elles puis sont activées afin qu'elles commencent à remplir leur fonction.

Le cycle de vie d'une application est tel que les opérations impliquées, qui normalement sont déclenchées séquentiellement, doivent être en partie réversibles pour rendre possible le retrait de l'application, sa désinstallation, sa désactivation ou son arrêt, et doivent pouvoir être déclenchées individuellement lors par exemple d'une mise à jour de l'application, de son retrait, ou de sa désactivation.

Nos travaux dans le cadre du projet Cubik se sont focalisés sur la mise à disposition de l'application (plus précisément sur les aspects liés à la spécification de l'assemblage de composants et de leur placement) et sur le lancement (plus précisément sur les aspects liés à l'instanciation des composants) en les appliquant au cas particulier des composants ubiquitaires. L'activité d'installation et plus précisément la problématique de transfert de paquetage a également fait l'objet de travaux dans l'équipe Casa. Nous aborderons ce point dans le paragraphe 2.3.7.

2.3.5.1 Déploiement de composants ubiquitaires, vue d'ensemble de l'approche Cubik

Spécification de placement Dans le cas d'une application distribuée, le déploiement induit un problème spécifique lié au placement des composants logiciels formant l'application. Il s'agit de définir, pour chaque instance de composant, la machine qui doit l'héberger. Ceci peut être fait lors de la description de l'assemblage des composants, en attribuant une machine cible à chaque composant spécifié, ou lors de l'installation, voire du lancement durant lequel on peut décider d'instancier ou pas un composant en fonction de critères évalués dynamiquement. Dans la plupart des technologies à composants, c'est durant la phase de mise à disposition, au moment du *packaging* des composants que l'on associe manuellement à chaque instance de composant une machine désignée typiquement par un nom ou une adresse IP. C'est le cas par exemple pour le déploiement de composants Corba [124] et des EJB [78], ou dans les travaux sur l'utilisation de Fractal pour le déploiement de serveurs J2EE [1].

Nous avons pour notre part choisi de ne pas contraindre la spécification de placement à faire figurer explicitement des noms de machine dans les descripteurs de déploiement. En effet, dans les réseaux dynamiques que nous avons considérés, la disponibilité de telle ou telle machine n'est pas forcément connue au moment du déploiement. Même lors du lancement, il se peut qu'une

machine inaccessible depuis un certain point du réseau devienne joignable rapidement du fait d'un changement de topologie. Nous avons au contraire autorisé une désignation des machines cibles du déploiement sous forme de contraintes sur les ressources, contraintes qui peuvent être évaluées à tout moment et que plusieurs machines peuvent satisfaire sans que ces machines soient connues à l'avance. Notre approche est similaire à celle décrite dans [40] qui propose de fonder le placement de composants logiciels sur la résolution d'un problème de programmation par contraintes.

Il faut noter que l'optimisation du placement n'était pas un de nos objectifs. Il apparaît que, dans un réseau relativement stable, il est possible de résoudre le problème de placement optimal des composants en fonction des ressources à la fois des machines et des capacités de liens entre les machines [112]. Mais nous voulions prendre en compte une dynamicité du réseau relativement forte, qui va au delà des hypothèses formulées dans les travaux pré-cités. Dans ces conditions, l'obtention d'un placement, même sous-optimal, demeure un problème difficile, et la recherche d'une solution optimale (qui resterait à définir précisément dans un contexte dynamique) n'est guère envisageable.

Infrastructure de déploiement La mise en œuvre du déploiement repose sur une infrastructure logicielle présente sur l'ensemble des machines cibles. L'organisation de cette infrastructure est traditionnellement centralisée et est exploitée via le paradigme client-serveur (cf. par exemple l'infrastructure de déploiement DCI [71] associée au support des composants Corba OpenCCM [18] ou le Fractal Deployment Framework [45]). Une exception notable implantant une approche distribuée du déploiement de composants Fractal est présentée dans [131]. Dans cette approche est construit un ensemble de contrôleurs de déploiement dont l'architecture suit celle de l'application. Nous avons adopté une démarche similaire dans le projet Cubik : les réseaux dynamiques, susceptibles d'être fragmentés, nécessitent une infrastructure de déploiement complètement décentralisée.

Autonomie Une grande partie des travaux de recherche sur le déploiement d'application à base de composants vise à automatiser le plus possible le processus de déploiement. Celui-ci reste toutefois souvent dépendant d'actions faites par un administrateur (ou un utilisateur averti) : identification de ressources cibles, prise en compte des contraintes architecturales, choix d'un appariement entre les composants et des machines cibles. Pour obtenir un processus autonome, ces tâches doivent être aussi automatisées. Ceci passe par une description des besoins en ressources des composants et une découverte des ressources des machines cibles qui permettent le calcul d'une solution de placement, ce calcul pouvant se solder par un échec si les ressources ne sont pas suffisantes [87, 13].

Notre objectif dans le cadre du projet Cubik allait au delà du placement initial des composants en considérant que la dynamicité du réseau pouvait faire varier la disponibilité des ressources et que le déploiement était donc éventuellement amené à remettre en cause des choix initiaux. Le processus de déploiement que nous avons mis en place est un processus continu et non pas une tâche aboutissant à un succès ou à un échec. Les composants sont lancés sur les machines hôtes au fur et à mesure de la découverte de la disponibilité de ressources. La capacité de fonctionnement en mode dégradé que nous avons décrite dans le paragraphe 2.3.4 permet d'activer une partie des composants même si tous les composants n'ont pas encore été lancés faute de ressources. De

même, l'impossibilité pour un composant de continuer à fonctionner du fait de la disparition d'une ressource se traduit (sous certaines conditions) par le fait qu'il est considéré comme non déployé, ce qui entraîne le processus de déploiement à chercher de nouveau une machine susceptible de l'héberger.

2.3.5.2 Spécification

Afin de pouvoir spécifier le déploiement de composants ubiquitaires Cubik, nous avons opté pour une approche déclarative. Nous avons proposé une extension de Fractal ADL ayant pour principal objet d'autoriser l'expression de contraintes posées sur les machines devant héberger les sous-composants. L'intergiciel de déploiement assure ensuite de manière autonome le lancement des composants sur les machines aptes à les accueillir.

La description des propriétés que les machines cibles doivent satisfaire est donnée dans un descripteur de déploiement dans lequel il est possible de faire figurer des références aux instances de composants qui sont définis dans le descripteur d'architecture. Pour chaque composant, un contexte de déploiement est ainsi spécifié, qui liste toutes les contraintes qu'une machine hôte doit satisfaire. Deux types de contraintes peuvent être employées dans un contexte de déploiement : les contraintes de ressources et les contraintes de localisation. Les contraintes de ressources servent à représenter les besoins matériels et logiciels des composants. Les contraintes de localisation sont utilisées pour guider le placement des composants (on contraindra par exemple deux composants à être placés sur des machines différentes).

Contraintes de ressources Pour définir une contrainte de ressource, il suffit d'associer à la définition d'un composant un contexte de déploiement qui liste l'ensemble des ressources nécessaires au composant. Nous nous sommes appuyés sur la modélisation des ressources de D-Raje décrite au paragraphe 2.2.4. On pourra donc par exemple imposer que la machine hébergeant un composant possède un minimum de 1 Go de mémoire vive, qu'elle dispose d'une version de la bibliothèque de traitement d'image *ImageMagick* supérieure à 6.2. De façon similaire, on pourra contraindre une liaison entre deux composants à être supportée par un lien physique caractérisé par un débit nominal d'au moins 100 Mbit/s.

Les contraintes citées ci-dessus sont bien sûr applicables aux composants primitifs. Mais nous avons choisi de permettre d'attacher des contraintes de ressources aussi aux composants composites. Ceci est justifié par le fait que si aucune contrainte n'était posée sur un composite, le déploiement sur une machine, commençant par le composite racine, pourrait progresser dans la hiérarchie en instanciant des membranes, même si au final aucun primitif n'est instanciable localement.

Dans certains cas, il est possible d'inférer des contraintes sur un composite à partir des contraintes posées sur les primitifs qu'il englobe. Par exemple, si un primitif p_1 requiert une bibliothèque logicielle b_1 et un autre primitif p_2 une bibliothèque logicielle b_2, il est clair que le composite englobant requiert b_1 et b_2. Toutefois, cet exemple est très difficilement généralisable à toutes les ressources. On peut par exemple imaginer que si p_1 requiert 1 Go de mémoire et que p_2 en requiert 2 Go, le composite englobant nécessite 3 Go (la somme) ou 2 Go (le maximum) selon la manière d'exploiter la mémoire. Notre approche ne fixe pas la façon d'établir les contraintes de ressources sur les composites, l'idée est que la fabrication d'un descripteur de déploiement

devrait être assistée par un outil propre au domaine d'application, qui établirait des règles par défaut de « composition » des contraintes.

Contraintes de localisation Afin de contraindre le placement de composants indépendamment de leurs besoins en ressources, il est possible d'inclure dans le descripteur de déploiement des contraintes dites de localisation attachées à un composant. Dans sa plus simple expression, une telle contrainte impose le placement du composant sur une machine désignée par son nom (ou son adresse IP). On peut également désigner la cible sous forme d'une variable libre qui pourra être utilisée dans d'autres contraintes. En effet, dans le cas général, une contrainte de localisation est une expression faisant intervenir des noms de machines et des variables représentant des machines cibles d'autres composants, combinés avec des opérateurs de comparaison n-aires. On pourra donc exprimer des contraintes comme « ce composant primitif doit être placé sur une machine différente de celle du composant serveur1 et celle du composant serveur2 » ou « ce composant composite a pour ensemble cible l'ensemble de machines comprenant la machine zeus et les machines m_1, m_2 et m_3, cibles du composant composite monComposant ».

Lorsque l'on considère la hiérarchie de composants Cubik, l'héritage des ensembles cibles est la règle par défaut. Donc, les variables libres spécifiées dans une contrainte de localisation se trouvent finalement liées à un nom de machine par la spécification de contraintes du composant qui l'englobe (directement ou pas). Il est possible de faire apparaître des variables libres au niveau du composant racine. Dans ce cas, la liste des noms de machines effectifs est supposée être passée en paramètre de l'ordre de déploiement.

Exemple La figure 2.12 montre un exemple de descripteur de déploiement de l'application de photo décrite dans la figure 2.9.

La première colonne contient les contraintes associées au composant composite DocumentSearch, telles qu'elles pourraient avoir été formulées par le fournisseur de ce composant acquis sur étagère pour bâtir l'application de photo. Ce descripteur contient le contexte de déploiement du composant primitif DocumentFinder (lignes 5 à 14) stipulant que ce composant nécessite un CPU cadencé à au moins 1,2 GHz et incluant une contrainte de localisation destinée simplement à nommer x la machine qui l'hébergera. De même, le contexte associé au composant primitif DocumentBuffer (lignes 17 à 26) pose une contrainte sur la mémoire de la machine qui peut l'héberger (au minimum 200 Mo), machine qui est nommée y. Enfin, le contexte de déploiement du composant DocumentSearch (lignes 29 à 36) impose, à travers une contrainte de localisation utilisant l'opérateur de différence *alldiff*, que la machine x soit différente de la machine y.

La deuxième colonne de la figure décrit les contraintes de déploiement du composant racine PhotoApp. On y retrouve des contraintes de ressources posées sur les composants primitifs DiapoMaker et PhotoRepository, notamment des contraintes sur la mémoire de stockage (espace disque) minimum requise dans le répertoire `/home`. On retrouve dans ce descripteur une contrainte de localisation sur le composant composite DocumentSearch (lignes 61 à 66) indiquant que les machines cibles pour ce composant ne peuvent être egilsay et parvati. La dernière partie du descripteur (lignes 70 à 76) donne une contrainte de localisation servant à décrire la plate-forme cible, à travers une liste de noms de machines.

```
<component name="DocumentSearch">
 <component name="DocumentFinder">
  <deployment-context>
   <resource-constraint>
   <cpu freq="1.2" unit="GHz"
      operator="min" />
   </resource-constraint>

   <location-constraint>
     <target varname="x"/>
   </location-constraint>
  </deployment-context>
 </component>

 <component name="DocumentBuffer">
  <deployment-context>
   <resource-constraint>
     <memory free="200" unit="MB"
          operator="min" />
   </resource-constraint>
   <location-constraint>
     <target varname="y"/>
   </location-constraint>
  </deployment-context>
 </component>

 <deployment-context>
  <location-constraint>
   <operator name="alldiff">
    <arg varname="this.DocumentFinder.x" />
    <arg varname="this.DocumentBuffer.y" />
   </operator>
  </location-constraint>
 </deployment-context>
</component>
```

```
<component name="PhotoApp">
 <component name="DiapoMaker">
  <deployment-context>
   <resource-constraint>
    <cpu freq="1.5" unit="GHz"
       operator="min" />
    <memory  free="50" unit="MB"
          directory="/home/"
          operator="min"/>
   </resource-constraint>
  </deployment-context>
 </component>

 <component name="PhotoRepository">
  <deployment-context>
   <resource-constraint>
    <memory free="1" unit="GB"
          directory="/home/"
          operator="min" />
   </resource-constraint>
  </deployment-context>
 </component>

 <component name="DocumentSearch">
   <locationconstraint>
    <operator name="exclude">
     <arg value="egilsay" />
     <arg value="parvati" />
    </operator>
   </locationconstraint>
 </component>

 <deployment-context>
  <locationconstraint>
   <target hostname="ambika"/>
   <target hostname="dakini"/>
   <target hostname="mafate"/>
   <target hostname="egilsay"/>
   <target hostname="parvati"/>
  </locationconstraint>
 <deployment-context>
</component>
```

Figure 2.12 – Descripteur de déploiement de l'application de photo

2.3.5.3 Processus de déploiement

Déploiement vu comme une résolution de CSP Le problème central à résoudre lors du déploiement d'un composant ubiquitaire Cubik est celui de l'instanciation de la hiérarchie de composants qu'il englobe. Il s'agit de maintenir une configuration valide du placement des instances de composants interconnectés qui respecte à la fois l'architecture du composant et l'ensemble de contraintes de ressources et de localisation qui lui sont attachées. Afin de permettre un déploiement autonome, nous avons choisi de considérer ce déploiement comme la résolution d'un problème de satisfaction de contraintes (ou CSP pour *Constraint Satisfaction Problem*). Les contraintes issues de l'architecture et du descripteur de déploiement sont réifiées et forment à l'exécution un CSP dont la résolution permet non seulement le déploiement mais aussi la reconfiguration autonome de l'application au gré de la fluctuation des ressources et de la mobilité des équipements du réseau. L'un des intérêts de cette approche est la possibilité de réutiliser des solveurs de contraintes éprouvés, à la fois en termes de robustesse et de performance.

Le déploiement est défini sous la forme d'un ensemble de CSP :

- CSPres, un CSP traduisant les contraintes de ressources présentes dans les descripteurs de déploiement ;
- CSPloc, un CSP traduisant les contraintes de localisation présentes dans les descripteurs de déploiement ;
- CSPinst, un CSP modélisant le fait qu'un composant primitif n'est instancié qu'une seule fois (la somme des instances sur les machines candidates est égale à 1) ;
- CSPbind, un CSP modélisant l'ensemble des liaisons entre composants.

Pour ces quatre ensembles on met en place des solveurs séparés. Ce découpage facilite en effet la distribution du processus de résolution. Les solveurs associés à CSPres et CSPinst ne posent pas de problème particulier de distribution dans la mesure où les processus de résolution sont indépendants d'une machine à l'autre. Comme nous le décrirons dans les paragraphes suivants, les solveurs CSPloc et CSPbind partagent des données qui doivent être maintenues cohérentes ; ils nécessitent donc d'être coordonnés.

Déploiement propagatif Le processus de déploiement vise à sélectionner une ou plusieurs machines pour chaque composant de l'application, conformément aux CSP définis plus haut. Il n'est pas envisageable de calculer un placement complet de façon centralisée. Le déploiement permet plutôt d'activer l'application progressivement, une partie des services de l'application étant disponible même si certaines machines nécessaires à certains composants non encore instanciés ne sont pas disponibles. Dès que ces machines le deviendront, le déploiement continuera. En outre, la progression du déploiement est assurée non seulement par le fait que certaines machines deviennent accessibles, mais aussi parce que des changements de ressources ont lieu qui rendent possible l'instanciation de composants. Nous qualifions ce processus de déploiement de propagatif pour évoquer le fait qu'un ordre de déploiement, issu d'une machine unique, provoque la propagation dans le réseau d'une vague d'activations de composants qui permet de rendre l'application progressivement opérationnelle.

La principale difficulté d'un tel processus est de garantir l'unicité des instanciations imposée par le descripteur d'architecture. Considérons l'ensemble des machines cibles d'un composant

composite. D'une part, une de ces machines ne peut être sélectionnée a priori pour héberger un sous-composant puisque cette machine peut ne pas être accessible. D'autre part, si chaque machine prend la décision d'héberger ou non un sous-composant, on ne peut garantir que dans des îlots différents où la coordination immédiate des machines est impossible, des décisions contradictoires ne seront pas prises, décisions qui conduiraient à des doublons d'instanciation.

Nous avons défini un algorithme distribué mettant en œuvre le déploiement propagatif tout en garantissant l'unicité des instanciations au sein d'un réseau dynamique potentiellement fragmenté en îlots. Nous en donnons ici les grandes lignes.

Quand le déploiement est lancé depuis une machine initiale, le descripteur de déploiement et le descripteur d'architecture sont diffusés à toutes les machines listées dans l'ensemble cible du composant racine. Chaque machine recevant ces descripteurs démarre un processus récursif (pour chaque composant composite) afin de sélectionner les sous-composants qui seront instanciés localement. Les principales étapes de ce processus sur la machine m_i, pour un composant C sont les suivantes :

1. m_i vérifie qu'elle appartient à l'ensemble cible de C. Si ce n'est pas le cas, le processus termine pour ce composant.
2. m_i lance des sondes correspondant aux contraintes de ressources de chaque sous-composant de C (par exemple une sonde pour observer la mémoire disponible). Pour chaque sous-composant pour lequel la sonde a retourné une valeur compatible avec la contrainte de ressource, m_i se déclare candidat à l'instanciation et diffuse sa candidature.
3. m_i reçoit aussi d'autres candidatures. Dès que m_i a calculé une solution en fonction de ces candidatures, elle tente de la faire adopter via un algorithme de consensus.
4. Une fois le consensus atteint, l'information de placement est envoyée aux autres machines (et donc aux autres candidats).
5. Le processus démarre de nouveau à l'étape 1 pour chaque sous-composant qui peut être instancié sur m_i.

Comme les ressources peuvent fluctuer, les sondes mises en place permettent une découverte périodique des ressources (point 2). En outre, il est possible qu'aucune solution de placement n'existe (point 3), c'est-à-dire qu'aucune combinaison de candidatures ne satisfasse les contraintes de localisation. Une observation périodique des ressources permet à une machine de candidater à l'instanciation d'un composant donné dès que les contraintes de ressources sont vérifiées, permettant l'émergence éventuelle d'une nouvelle solution des contraintes de localisation.

Au point 3, l'information de placement est diffusée aux autres machines. Ceci permet de globalement mettre à jour le descripteur de déploiement avec les nouvelles valeurs pour les machines cibles, en complétant la désignation de variable par un nom de machine effectif. Par exemple, si les machines ambika et dakini sont choisies respectivement pour les composants DocumenFinder et DocumentBuffer, les changements suivants seront opérés sur le descripteur donné à la figure 2.12 :

ligne 12 remplacée par <target varname="x" value="ambika"/>
ligne 24 remplacée par <target varname="y" value="dakini"/>

Le descripteur étant persistant, la progression du déploiement sera maintenue même après un arrêt de machine.

Déploiement autonome Le déploiement propagatif permet aux composants ubiquitaires d'être déployés dès que les ressources qu'ils requièrent sont disponibles. Mais en général, et particulièrement dans les réseaux dynamiques, les ressources peuvent venir à manquer (par exemple la quantité de mémoire disponible peut diminuer et devenir insuffisante) et des fautes peuvent survenir. Dans de tels cas, un ou plusieurs composants doivent pouvoir être re-déployés. Ce redéploiement se passe en trois temps :

1. Chacun des composants qui dépendent d'une ressource non disponible est stoppé, ce qui provoque la désactivation de ses interfaces fournies et, de proche en proche, de toutes les interfaces locales et distantes qui en dépendent.
2. L'état du composant est sauvegardé dans une forme sérialisée. On suppose que le programmeur du composant a anticipé cette situation en implantant une méthode adéquate accessible via une interface de contrôle.
3. Un message contenant l'identité du composant à redéployer est diffusé. Ce message indique aussi la localisation du composant sérialisé. Chaque machine recevant ce message met à jour son descripteur de déploiement en supprimant la localisation effective du composant concerné.

En plus des changements de ressources, les pannes de composants sont une cause de redéploiement. Nous avons fait l'hypothèse que seuls les composants pouvaient subir des pannes franches, les pannes de machines et de l'intergiciel lui-même étant écartées.

La procédure décrite ci-dessus suffit à définir un déploiement autonome. En effet, quand elles reçoivent le message diffusé au point 3, les machines – parce qu'elles ont mis à jour leur descripteur de déploiement – se retrouvent de nouveau dans le processus de déploiement propagatif : des composants sont encore à déployer. Ainsi, puisque le déploiement n'est pas complètement terminé, le processus reste actif, des machines candidateront pour l'instanciation du composant non encore installé.

Consensus Le processus distribué de déploiement décrit plus haut implique une prise de décision collective cohérente pour choisir la machine qui instanciera un composant donné, sachant que, potentiellement, les conditions de ressources sont vérifiées sur plusieurs machines et qu'il peut exister des contraintes de co-localisation de composants.

Dans un système asynchrone sujet à défaillance, il a été démontré qu'il n'existe pas d'algorithme déterministe permettant de résoudre le problème de consensus [52]. Néanmoins, un affaiblissement du problème, ou plus exactement une restriction de l'espace de définition du consensus, permet sa résolution. Nous avons adapté un algorithme de consensus distribué décrit dans [114] pour permettre d'élire parmi un ensemble de candidats la machine dont l'identité sera approuvée par une majorité de machines. Les auteurs de cet algorithme ont identifié des conditions dans lesquelles il existe un protocole asynchrone résolvant le problème de consensus malgré l'occurrence de t processus fautifs. Dans notre cas, s'il y a n machines impliquées dans le déploiement, t peut atteindre $\left\lfloor \frac{n}{2} \right\rfloor$. Ainsi, une prise de décision collective peut être faite dans un îlot contenant une majorité des machines. En nous appuyant sur la majorité, nous garantissons qu'au sein de l'îlot, il y au moins une machine qui possède la dernière version du descripteur de déploiement et, ainsi, qu'aucune décision contradictoire ne peut être prise dans deux îlots distincts.

Notre algorithme de consensus nécessite que le nombre de machines accessibles parmi les machines cibles d'un composant composite atteigne la majorité. Il faut noter que cette majorité n'est

pas la même pour tous les composants. Par exemple, le composant racine de notre application de photo est ubiquitaire sur $\{m_1, m_2, m_3, m_4, m_5\}$; en conséquence, la majorité est atteinte pour ce composant quand au moins trois machines sont présentes dans le même îlot. En revanche, pour le composant DocumentSearch, ubiquitaire sur $\{m_1, m_2, m_3\}$, le consensus est résolu quand un îlot composé d'au moins $\{m_1, m_2\}$, $\{m_1, m_3\}$ ou $\{m_2, m_3\}$ est formé, soit un îlot de deux machines.

Le nombre de machines dans un îlot peut ne pas être suffisant pour faire aboutir le consensus. Pour que le consensus ne soit pas bloqué indéfiniment, une machine nouvellement accessible est autorisée à y participer. Ceci est réalisé grâce à des diffusions de requêtes pour déterminer les consensus en cours. Une machine recevant une réponse positive, et détectant donc un consensus en cours, peut collecter les données déjà échangées entre les autres machines et proposer une valeur qui peut faire évoluer le consensus.

La panne d'un composant est détectée sur la machine qui l'héberge. Celle-ci met à jour son descripteur de déploiement en enlevant la localisation en cours pour le composant. L'algorithme propagatif propagera cette information pour rechercher une autre machine candidate à l'instanciation du composant.

Le protocole décrit dans [114] repose sur des communications fiables. Le type de réseau dynamique que nous visons ne nous permet pas de garantir cette fiabilité. À l'instar de ce qui a été proposé dans [103], un composant de retransmission périodique a été ajouté et un système d'accusés de réception mis en place pour fiabiliser les communications. Pour éviter une surcharge du réseau, le composant de retransmission a été couplé à un mécanisme de détection du voisinage.

2.3.6 Mise en œuvre

Un prototype d'intergiciel supportant les composants ubiquitaires Cubik a été mis en œuvre. À partir d'une description d'architecture de composants Fractal et d'un descripteur de déploiement écrit dans le langage de spécification décrit au paragraphe 2.3.5.2, cet intergiciel réalise les fonctions suivantes :

- le déploiement initial de l'application qui inclut l'instanciation des composants primitifs, des membranes et des composants mandataires ;
- l'activation et la désactivation des interfaces au gré de la fluctuation de l'accessibilité des composants ;
- le redéploiement dynamique des composants en cas de pannes de composants ou de violation des contraintes de ressources.

Le prototype a lui-même été réalisé comme une application à composants Fractal et s'appuie sur l'implantation de référence Julia. Son architecture générale est représentée dans la figure 2.13. On y retrouve les principaux services décrit dans les paragraphes précédents :

- Le composant Saje correspond à l'encapsulation dans un composant de la bibliothèque Saje, sous-ensemble centralisé de l'intergiciel d'observation de ressources D-Raje mentionné au début de ce chapitre. Ce composant permet l'observation périodique des ressources système qui alimente la vérification des contraintes de ressources associées aux composants. En plus des ressources système, des ressources de nature applicative ont été modélisées dans Saje : il s'agit notamment des liaisons entre composants ou des composants eux-mêmes. Ces ressources font l'objet d'une observation à la base de la gestion des interfaces actives.

- Le composant Communication P2P implante les communications asynchrones entre les différentes instances de l'intergiciel présentes sur les machines du réseau. Il est bâti au dessus d'UDP et incorpore des fonctions de fiabilisation.
- Le composant Consensus implante l'algorithme de décision collective d'instanciation des composants.
- Le composant Résolveur assure la résolution des contraintes de ressources, de localisation, d'instanciation et de liaisons. Il s'appuie sur la bibliothèque Cream [3], une bibliothèque Java d'outils pour les applications de satisfaction de contraintes ou d'optimisation dans des domaines finis. Une transcription des contraintes Cubik est réalisée dynamiquement pour se ramener à des contraintes sur les entiers.
  Nos doutes sur les performances de la résolution de contraintes ont été levés par une série d'expériences. En particulier, nous avons évalué la résolution de contraintes faisant intervenir l'opérateur le plus gourmand en calcul (opérateur alldiff, de complexité $O(n^2)$) sur une application impliquant jusqu'à une centaine de composants. Les temps obtenus sur des ordinateurs portables standard (Pentium Centrino 1,7 GHz) étaient de l'ordre de 20 à 30 ms, ce qui reste tout à fait praticable compte tenu des temps de transmission de messages qu'il faut par ailleurs supporter dans les réseaux dynamiques visés.
- Le composant CubikADL fournit les utilitaires de manipulation des descripteurs de déploiement. Il s'appuie sur l'usine Fractal ADL, ensemble extensible de modules permettant de traiter les définitions faites à l'aide du dialecte XML Fractal ADL.
- Le composant Déployeur joue le rôle de « chef d'orchestre ». Il coordonne le processus de déploiement propagatif et autonome. Il assure par ailleurs la mise en œuvre des liaisons distantes entre composants. Ces liaisons sont assurées par des mandataires pour lesquels des talons et squelettes sont mis en place grâce à Fractal RMI [4].

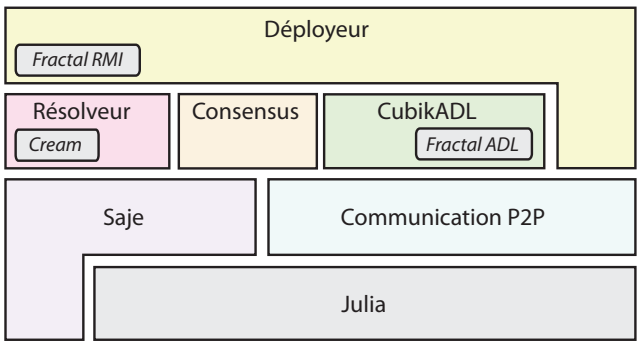


Figure 2.13 – Architecture générale du support des composants ubiquitaires

2.3.7 Bilan

Nos travaux sur les composants ubiquitaires ont pour l'essentiel été menés dans le cadre de la thèse de Didier Hoareau, entre 2003 et 2007. Les résultats principaux ont été les suivants.

3. http://bach.istc.kobe-u.ac.jp
4. http://fractal.objectweb.org/fractalrmi

– *Définition d'un modèle de composants ubiquitaires*
  Nous avons défini un mode de distribution de composants hiérarchiques dans lequel les composants sont rendus ubiquitaires : les interfaces des composants composites sont accessibles sur l'ensemble des machines du réseau tandis que chaque composant primitif, renfermant du code, est localisé sur une seule machine. Le réseau cible n'étant pas connexe, certaines interfaces peuvent, temporairement, ne plus être invocables. Par conséquent, un composant peut être amené à fonctionner en mode dégradé, offrant seulement une partie de ses fonctions. Le modèle de composants Fractal a été étendu pour permettre de manipuler l'état d'une interface (actif ou inactif). L'intergiciel assurant l'exécution distribuée des composants détecte les déconnexions et reconnexions sur un lien réseau et reflète ces événements sur l'état des interfaces supportées par ce lien.

– *Spécification du déploiement*
  Nous avons étendu le langage de description d'architecture de Fractal pour inclure la spécification du déploiement des composants. Cette spécification inclut des propriétés sur les ressources que doivent vérifier les machines pour pouvoir héberger un composant donné. On peut ainsi imposer des contraintes sur les ressources nécessaires aux composants et sur la localisation des composants. L'intergiciel supportant le déploiement des composants exploite cette information pour choisir automatiquement les machines sur lesquelles seront instanciés les composants.

– *Déploiement propagatif*
  L'intergiciel Cubik réalise un déploiement progressif de l'application : les composants sont instanciés au fur et à mesure de l'apparition des ressources disponibles. Grâce à la possibilité d'activer une partie des interface des composants, l'application peut être démarrée dès le début du déploiement, de nouvelles interfaces étant progressivement activées, lors de l'activation de nouveaux composants. Le passage à l'échelle est facilité par l'organisation hiérarchique du contrôle du processus de déploiement.

Ces résultats ont fait l'objet des publications suivantes :

– Didier Hoareau, Yves Mahéo. Middleware Support for Ubiquitous Software Components. Personal and Ubiquitous Computing (PUC), 12(2):167–178, février 2008, Springer [**j3**].

– Didier Hoareau, Takoua Abdellatif, Yves Mahéo. Architecture-Based Autonomic Deployment of J2EE Systems in Grids. *In Proceedings of the International Conference on Grid and Pervasive Computing (GPC'07)*, Paris, France, mai 2007, LNCS, volume 4459, pages 362–373, Springer [**i6**].

– Takoua Abdellatif, Didier Hoareau, Yves Mahéo. Automated Deployment of Enterprise Systems in Large-Scale Environments. *In Proceedings of the 8th International Symposium on Distributed Objects and Applications (DOA'06)*, Montpellier, France, novembre 2006, LNCS, volume 4277, pages 30–31, Springer [**i7**].

– Didier Hoareau, Yves Mahéo. Ubiquitous Fractal Components. *In Proceedings of the 5th Fractal Workshop, 20th European Conference on Object-Oriented Programming (ECOOP 2006)*, Nantes, France, juillet 2006 [**i9**].

– Didier Hoareau, Yves Mahéo. Constraint-Based Deployment of Distributed Components in a Dynamic Network. *In Architecture of Computing Systems (ARCS 2006)*, Frankfurt/Main, Allemagne, mars 2006, LNCS, volume 3864, pages 450–464, Springer [**i8**].

- Didier Hoareau, Yves Mahéo. Propagative Deployment of Hierarchical Components in a Dynamic Network. *In Proceedings of the 3rd International Working Conference on Component Deployment (CD 2005)*, Grenoble, France, novembre 2005, LNCS, volume 3798, pages 111–114, Springer [**i10**].
- Didier Hoareau, Yves Mahéo. Distribution of a Hierarchical Component in a Non-Connected Environment. *In Proceedings of the 31st Euromicro Conference – Component-Based Software Engineering Track*, Porto, Portugal, septembre 2005, pages 143–150, IEEE CS [**i11**].
- Didier Hoareau, Yves Mahéo. Distribution d'un composant hiérarchique dans un environnement partiellement connecté. *In Actes des Journées Composants 2005*, pages 103–111, Le Croisic, France, avril 2005 [**n1**].

L'étude du déploiement menée dans le cadre du projet Cubik s'est focalisée d'une part sur la spécification du déploiement et d'autre part sur le lancement des composants (et plus spécifiquement sur l'instanciation des composants). La phase d'installation a été volontairement ignorée, l'hypothèse étant faite que le code du composant à instancier était disponible sur la machine concernée. Dans un univers connecté, cette hypothèse est fondée dans la mesure où un téléchargement à la volée du code depuis un entrepôt donné est possible. Dans le cas des réseaux dynamiques, le problème de l'accessibilité du code peut se poser. Cet aspect du déploiement a été étudié dans l'équipe Casa au cours d'un autre projet : le projet MASC (*Mobile Software Adaptive Components*), projet financé par la région Bretagne entre 2002 et 2005.

Le projet MASC portait sur le déploiement d'applications à composants dans un environnement ubiquitaire prenant la forme d'un réseau mobile ad hoc sans connectivité de bout en bout. Dans ce projet, contrairement à la vision en vigueur dans le projet Cubik, les applications ne sont pas distribuées à proprement parler : une application s'exécute sur une machine unique et par conséquent, les codes des composants qui la constituent doivent être tous rapatriés sur cette machine pour pouvoir lancer l'application. La problématique étudiée au cours du projet MASC était celle de l'installation, sur une machine, de paquetages logiciels, un paquetage embarquant le code d'un composant et dépendant éventuellement d'autres paquetages. Un intergiciel nommé Codewan a été développé afin de permettre ce déploiement au sein d'un réseau mobile ad hoc sans connectivité de bout en bout. Le modèle de déploiement implanté dans Codewan est un modèle coopératif pair à pair : chaque nœud du réseau maintient un entrepôt local de paquetages et cherche à le remplir de manière opportuniste en fonction de ses besoins. Au gré de ses rencontres avec les autres nœuds du réseau, il peut en effet obtenir les paquetages qui lui manquent. Une couche de communication implante les primitives de diffusion et de transmission point à point qui sont à la base des interactions entre nœuds voisins.

Nos travaux sur le déploiement dans le cadre du projet MASC se sont soldés par la publication de l'article suivant :

- Frédéric Guidec, Nicolas Le Sommer, Yves Mahéo. Opportunistic Software Deployment in Disconnected Mobile Ad Hoc Networks. *International Journal of Handheld Computing Research (IJHCR)*, 1(1) :24–42, janvier 2010, IGI Publishing [**j2**].

3

Approche orientée services pour le développement d'applications sur réseaux mobiles ad hoc discontinus

3.1 Introduction

L'approche orientée services prend une importance grandissante dans le domaine du développement d'applications distribuées. Dans cette approche, les applications sont construites à partir du concept de service logiciel, assez proche du concept de composant logiciel dans la mesure où il implique une séparation claire entre interface et implantation. L'objectif est notamment de faciliter un développement et un déploiement indépendant des fournisseurs et clients de services constituant l'application. L'un des intérêts majeurs de cette approche réside en effet dans le découplage entre clients et fournisseurs. Ce découplage se manifeste non seulement lors de la conception de l'application mais aussi lors de son exécution. Les liaisons entre les clients et les fournisseurs sont effectuées tardivement : pendant l'exécution, une phase de découverte permet aux clients d'obtenir des informations relatives aux services disponibles et aux fournisseurs de ces services. Un client établit une liaison avec un fournisseur dynamiquement et cette liaison peut être rompue (par le client ou le fournisseur, ou du fait de la modification des conditions de connectivité dans le réseau) et rétablie plus tard, éventuellement avec un autre fournisseur.

Les caractéristiques de l'approche orientée services ont permis de faire émerger une offre commerciale de « briques de base » d'applications distribuées sur Internet, essentiellement à travers les services Web. Nous nous sommes intéressés pour notre part au paradigme de la programmation orientée services parce qu'il offre un certain nombre d'atouts pour le développement d'applications visant des réseaux dynamiques. La phase de découverte peut faciliter la prise en compte de l'hétérogénéité des nœuds et des liens de ces réseaux : les clients sont à même de sélectionner les services et fournisseurs en fonction des caractéristiques exposées, qu'elles soient

de nature fonctionnelle ou non fonctionnelle (relevant notamment de la qualité de service). En outre, le découplage à l'exécution des entités de l'application aide à supporter les changements intervenant dans le réseau du fait de la volatilité ou de la mobilité des nœuds.

À travers nos travaux sur les composants ubiquitaires, décrits au chapitre 2, nous avons constaté la difficulté inhérente à l'approche orientée composants de prendre en compte des applications devant s'exécuter sur un réseau très dynamique : l'assemblage des composants de l'application est dicté a priori et cette information, intrinsèquement globale, ne peut pas être aisément maintenue cohérente au sein d'un réseau qui ne présente pas une connectivité permanente. Ceci tend à limiter cette approche à des réseaux peu mobiles et relativement fermés, dont on connaît la composition. L'approche orientée services n'impose pas de décrire a priori un assemblage des entités logicielles interagissant au sein de l'application [1], les liaisons étant établies dynamiquement à l'initiative des clients de services. Elle paraît donc plus adaptée aux réseaux très dynamiques.

L'objectif des travaux qui sont décrits dans ce chapitre était d'explorer l'utilisation de l'approche orientée services pour développer et exécuter des applications visant des réseaux particulièrement dynamiques et ouverts. Nous avons choisi comme réseaux cibles les réseaux mobiles ad hoc (ou MANET), c'est-à-dire des réseaux formés spontanément par des équipements mobiles communiquant directement entre eux, sans passer par une infrastructure fixe. Nous nous sommes focalisés sur la classe des MANET discontinus, MANET dans lesquels la connectivité de bout en bout n'est pas garantie. Ces réseaux posent des problèmes de communications étudiés depuis moins d'une dizaine d'années. Les travaux de l'équipe Casa ont porté à la fois sur les problématiques de communication et sur les problématiques de fourniture de services, très liées dans le cas des MANET discontinus.

Le reste de ce chapitre est organisé comme suit. Le paragraphe 3.2 présente les caractéristiques des MANET discontinus et les problématiques de communication afférentes. Dans le paragraphe 3.3, on fait un rapide survol de l'approche orientée services et des plates-formes logicielles supportant cette approche pour les MANET. Le paragraphe 3.4 détaille les différentes contributions que nous avons apportées en ce qui concerne la communication et la fourniture de services dans les MANET discontinus.

3.2 Réseaux mobiles ad hoc discontinus

3.2.1 MANET connexes et MANET discontinus

Réseaux mobiles d'infrastructure et réseaux mobiles ad hoc Les applications distribuées ciblant des terminaux mobiles s'appuient le plus souvent sur des réseaux dits « d'infrastructure ». En exploitant des technologies de la téléphonie cellulaire (GSM, GPRS, UMTS…) ou plus récemment des normes comme IEEE 802.11 [75] (Wi-Fi) ou IEEE 802.16 [77] (WiMAX), les équipements mobiles de ces réseaux accèdent, via une liaison radio, à un équipement fixe (antenne relais GSM, point d'accès Wi-Fi…) servant de passerelle vers un réseau filaire, le plus souvent vers Internet. L'avantage principal de ce type d'architecture est de permettre à chaque terminal

1. L'approche orientée services n'exclut toutefois pas une description, faite lors de la conception, des interactions prévues entre clients et fournisseurs. C'est par exemple ce qui est visé avec l'utilisation des orchestrations et des chorégraphies dans le cadre des services Web.

un accès potentiel à toutes les ressources d'Internet. Mais une de ses conséquences directes est que deux terminaux mobiles doivent forcément passer par l'infrastructure pour communiquer. Le prix à payer est donc la dépendance vis-à-vis d'une infrastructure lourde et coûteuse dont l'administration est souvent très complexe.

Les réseaux mobiles ad hoc (ou MANET pour *Mobile Ad hoc NETworks*) représentent une forme alternative de réseaux dans laquelle les équipements mobiles s'auto-organisent pour former un réseau de manière spontanée. Les communications se font directement entre deux terminaux mobiles à portée radio. Des technologies de communication comme Wi-Fi (dans le mode ad hoc), ou Bluetooth (IEEE 802.15.1 [76]) servent typiquement de support pour la communication entre deux terminaux. Dans un MANET, tous les nœuds peuvent jouer peu ou prou le même rôle, et chacun d'eux peut participer à l'acheminement des données à travers le réseau, permettant ainsi de se passer de toute infrastructure fixe.

Un réseau mobile ad hoc n'est pas forcément isolé. Dans le cas où un ou plusieurs équipements d'un MANET disposent d'un accès à un réseau d'infrastructure, on peut exploiter le MANET comme une extension de ce réseau d'infrastructure.

Typologie des réseaux mobiles ad hoc Les équipements formant un MANET peuvent prendre des formes très variées. Dans un cas que l'on peut considérer comme standard, il s'agit d'ordinateurs portés par des individus et qui suivent donc ces individus dans leurs déplacements. Toute une gamme d'ordinateurs peut être utilisée, de l'ordinateur portable au téléphone intelligent (*smartphone*), en passant par le mini-PC (*netbook*) ou l'assistant numérique personnel (PDA pour *Personal Digital Assistant*). Outre la mobilité, un facteur clé à prendre en compte lors du développement des protocoles de communication et des applications est l'autonomie énergétique relativement faible des équipements.

Les réseaux véhiculaires (VANET pour *Vehicular Ad hoc NETtworks*) forment une famille particulière de MANET dans laquelle les équipements communicants sont embarqués dans des véhicules, ce qui leur confère notamment des capacités accrues en terme d'énergie, de calcul et de stockage ainsi que des caractéristiques de mobilité propres [125]. Les communications se font entre véhicules ou entre les véhicules et des dispositifs fixes présents sur les bords de la route (par exemple des panneaux de signalisation). L'échange de données peut permettre d'informer les usagers des conditions de circulation, de partager du contenu multimédia, de coordonner le déplacement des véhicules, etc.

Les réseaux de capteurs peuvent former des MANET lorsque les capteurs communicants sont mobiles, portés par des humains ou des animaux, embarqués dans des véhicules ou ayant eux-mêmes des capacités de mouvement. L'objectif est typiquement la collecte de données enregistrées par les capteurs (température, vitesse, paramètres biologiques…) et leur acheminement jusqu'à un équipement capable de les traiter. La contrainte énergétique est souvent très forte du fait de la petite taille des capteurs.

Dans tous les cas, la capacité de géolocalisation des équipements mobiles a une grande influence non seulement au niveau applicatif mais aussi au niveau des protocoles mis en place dans les couches plus basses pour assurer un acheminement efficace des messages dans le réseau.

Applications des réseaux mobiles ad hoc Les applications des réseaux mobiles ad hoc ont été dans un premier temps envisagées dans le domaine militaire, dans lequel la capacité

d'auto-organisation est un atout pour faire face aux destructions éventuelles de matériel. Depuis quelques années un certain nombre d'applications civiles sont également étudiées.

Le domaine de l'intervention en cas de crise (*disaster relief*) est un domaine qui retient une attention croissante. Après une catastrophe naturelle (tremblement de terre, inondation…) entraînant la destruction partielle ou totale des équipements de communication fixes, il s'agit d'être capable de permettre aux victimes de contacter les autorités et d'assurer la coordination des équipes de secours. De façon plus générale, l'utilisation des réseaux mobiles ad hoc est évidemment pertinente lorsqu'on ne peut pas disposer d'une infrastructure fixe – tout du moins à un coût raisonnable – comme par exemple lors d'expériences visant à collecter des données sur les déplacements d'animaux grâce à des capteurs [150, 80, 137], ou pour offrir des moyens de communication dans des zones rurales [130, 44]. Mais les MANET peuvent aussi être utilisés comme alternative à des réseaux d'infrastructure pourtant présents, pour des raisons de coût ou de nature sociale (on parle par exemple de communication communautaire ou de « tribu » dans le cas d'un ensemble d'individus échangeant des données uniquement entre eux, sans passer par un opérateur de réseau).

Routage dans les réseaux mobiles ad hoc Une grande partie des travaux de recherche effectués depuis une quinzaine d'années dans le domaine des MANET a porté sur le développement de méthodes d'acheminement des messages (essentiellement des paquets IP) de bout en bout, en s'appuyant sur les communications entre nœuds voisins. L'objectif de ces travaux est de définir des protocoles de routage IP exploitant chacun des équipements du réseau comme routeur, et supportant les changements de topologie du réseau dus à la mobilité des nœuds. On parle de MANET multi-sauts pour désigner un MANET doté d'un tel protocole de routage [2].

De nombreux protocoles de routage IP adaptés aux MANET ont été développés [134, 98]. Le groupe de travail MANET de l'IETF (*Internet Engineering Task Force*) a entrepris de recenser et évaluer ces divers protocoles en vue d'une normalisation. Deux principaux types de routage se dégagent, le routage pro-actif et le routage réactif, distinguant les protocoles selon leur façon de constituer la table de routage présente sur chaque nœud.

Dans le routage pro-actif (*proactive routing* ou *table-driven routing*), chaque nœud s'efforce de maintenir constamment à jour sa propre table de routage, de sorte que lorsqu'un paquet IP doit être émis par ce nœud, la route que ce paquet doit suivre soit d'ores et déjà connue, et donc immédiatement exploitable. Pour ce faire, chaque nœud diffuse périodiquement sa propre table de routage, et la met par ailleurs à jour en fonction d'informations similaires reçues de tous ses voisins. Les avantages majeurs présentés par les protocoles de routage pro-actifs sont les bonnes performances qu'ils permettent d'obtenir lorsqu'un paquet IP doit suivre une route dans laquelle tous les nœuds-relais sont effectivement disponibles et disposent chacun d'une table de routage à jour. Les inconvénients majeurs sont le surcoût important occasionné par le trafic de contrôle nécessaire au maintien à jour des informations de routage, et des temps de réaction souvent assez longs lorsque des changements dans la topologie du réseau obligent l'ensemble des hôtes à corriger leurs tables de routage en conséquence. Le protocole OLSRv2 (*Optimized Link-State Routing protocol* [33]) fait office de protocole pro-actif de référence.

Dans le routage réactif (*reactive routing* ou *on-demand routing*), les tables de routage ne sont

2. Il faut noter que, très fréquemment, on a tendance à considérer qu'un MANET est par définition doté d'une capacité de communication de bout en bout fondée sur une forme de routage IP.

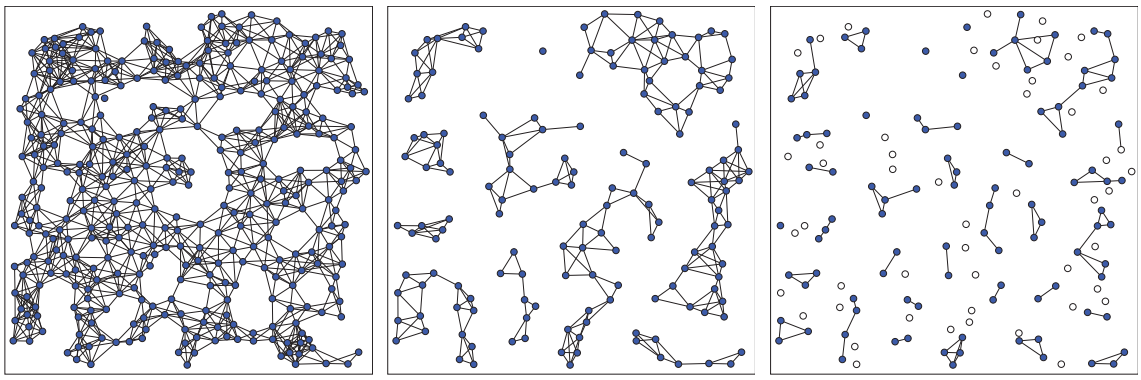
mises à jour que lorsque le besoin s'en fait sentir. Lorsqu'un nœud devant router un paquet IP vers une certaine destination constate que sa table de routage ne contient aucune indication pour atteindre cette destination, il diffuse dans l'ensemble du réseau un message de contrôle invitant tous les nœuds du réseau à mettre à jour leurs tables de routage vis-à-vis de la destination visée. En règle générale, cette mise à jour s'effectue grâce à un second message de contrôle initié par l'hôte destinataire lui-même, ce message remontant (en *source-routing*) vers le nœud qui cherche à l'atteindre. Alternativement, une diffusion peut être utilisée pour inciter tous les nœuds à mettre à jour dans leur table de routage les routes menant vers ce destinataire précis. L'avantage majeur de l'approche réactive est que le surcoût occasionné par la mise à jour des informations de routage est directement proportionné, au cas par cas, aux flux de données circulant dans le réseau. Le principal inconvénient réside dans l'important temps de latence nécessaire pour qu'une route vers une certaine destination puisse être exploitée pour la première fois (ou encore en cas de changements de topologie fréquents, obligeant à une réactualisation fréquente des routes dans le réseau). La protocole DYMO (*DYnamic Manet On-demand Routing* [27]) est en passe d'être standardisé par l'IETF en tant que protocole réactif de référence.

D'autres protocoles dits hybrides combinent les approches pro-actives et réactives en partitionnant l'ensemble des nœuds du réseau en grappes et en différenciant la technique de routage utilisée au sein des grappes et entre les grappes.

Par ailleurs, certains protocoles dits de routage géographique supposent que les nœuds sont capables de se géolocaliser, c'est-à-dire d'estimer de façon plus ou moins précise leur position relative ou absolue dans le réseau. À partir des informations de localisation, de tels protocoles peuvent évaluer les distances entre nœuds et la distribution des nœuds dans l'espace afin d'optimiser l'algorithme de routage.

Réseaux mobiles ad hoc discontinus Les approches fondées sur le routage dynamique supposent que deux équipements quelconques du réseau peuvent communiquer via une route établie dynamiquement entres ces deux nœuds grâce à un ensemble de nœuds intermédiaires. Pour qu'une transmission puisse avoir lieu de bout en bout, il faut donc que l'ensemble de ces nœuds intermédiaires soient disponibles pour assurer le routage. Si un de ces nœuds devient inaccessible, même temporairement, le paquet IP est détruit en chemin. Faire reposer les applications distribuées sur un tel routage suppose en réalité que le réseau mobile ad hoc ait une densité et une distribution spatiale des nœuds favorables, de sorte qu'il soit en permanence connexe. La figure 3.1-(a) illustre un tel réseau de 350 terminaux mobiles répartis dans une zone de 1 km × 1 km. Une arête entre deux nœuds représente la capacité de communication directe entre deux terminaux, dépendant essentiellement de la portée des transmissions radio. La densité est suffisante pour assurer une connexité quasi-permanente du réseau, avec des schémas de mobilité très variés. C'est un exemple typique de MANET dans lequel on pourra exploiter les protocoles de routage IP décrits plus haut.

Cependant, dans un grand nombre de situations, les caractéristiques de densité, de distribution et de mobilité des nœuds du réseau ne peuvent garantir la connexité du réseau. Dans la partie (b) de la figure 3.1, seuls 150 nœuds sont présents. On constate alors qu'il n'est plus possible de trouver un chemin de bout en bout pour toute paire de terminaux mobiles. Le graphe est partitionné en plusieurs « îlots » au sein desquels les communications sont possibles (en utilisant éventuellement du routage dynamique). Mais aucune communication n'est a priori possible entre des îlots distincts. Les possibilités de communication sont globalement encore plus réduites



(a) MANET connexe (b) MANET discontinu sans volatilité (c) MANET discontinu avec volatilité

Figure 3.1 – MANET connexes et discontinus

si on tient compte de la volatilité des terminaux mobiles, c'est-à-dire du fait qu'ils peuvent être parfois éteints (ou mis en veille) et donc temporairement dans l'incapacité de communiquer. Dans la partie (c) de la figure 3.1, un terminal sur trois est considéré comme éteint (cercles vides), ce qui augmente significativement la fragmentation du réseau.

On parle de MANET discontinus (*disconnected MANETs*) pour désigner des MANET potentiellement fragmentés en îlots, par opposition au MANET connexes (*connected MANETs*) dans lesquels le routage IP dynamique peut être utilisé.

3.2.2 Communication dans les MANET discontinus

Communication tolérant les délais et communication opportuniste Les travaux de recherche sur la communication dans les MANET discontinus ont été initiés il y a peu d'années mais connaissent maintenant un essor certain. De façon générale, ces travaux visent à remplacer ou compléter les mécanismes de routage dynamique conçus pour les MANET connexes par des mécanismes permettant de tolérer les ruptures de connectivité occasionnelles ou chroniques.

L'approche communément adoptée consiste à doter tout ou partie des terminaux mobiles de la capacité de stocker temporairement des messages dans un cache avant de les réémettre au moment opportun. Ce stockage temporaire permet non seulement de supporter une incapacité du terminal à transférer utilement le message à un autre terminal immédiatement, mais aussi de profiter du déplacement d'un porteur de message pour transporter physiquement ce message vers une autre partie du réseau avant de le transmettre. Ce modèle de comportement des terminaux est désigné sous le vocable *store, carry and forward*. Il est illustré dans la figure 3.2 qui présente la transmission d'un message, en cinq étapes successives, dans un MANET discontinu initialement fragmenté en trois îlots. Le terminal A souhaite transmettre un message au terminal H. Initialement, A n'a pas de voisin à portée radio et se contente de conserver son message. Puis (étape 2) il s'approche des terminaux B, C et D et choisit de transmettre son message à C (étape 3). Le terminal C stocke le message dans son cache. Ultérieurement, C se déplace lui-même et se rapproche de G (étape 4). Enfin, en s'appuyant sur du routage dynamique, C transmet le message à H par l'intermédiaire de G et de E.

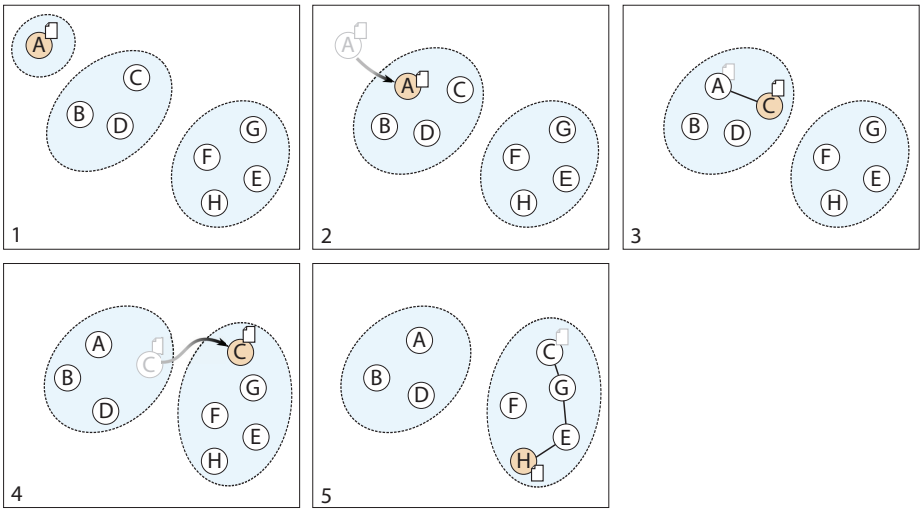


Figure 3.2 – Illustration du principe de *store, carry and forward*

L'application du principe du *store, carry and forward* est apparu dans les travaux relevant de la communication dans les réseaux « tolérant les délais » (DTN pour *Delay-Tolerant Networking*). Le DTNRG (*Delay-Tolerant Networking Working Group* [3]) de l'IRTF (*Internet Research Task Force*) fédère un ensemble de recherches sur des architectures de réseaux sans connectivité de bout en bout. Le premier type de réseaux visé était celui des réseaux interplanétaires dans lesquels les possibilités de communication sont intermittentes (du fait par exemple de l'attente nécessaire du passage d'un satellite) et sujettes à d'importants délais dus aux distances entre les équipements communicants. Plus récemment, il s'est avéré que le champ d'application du DTN pouvait couvrir aussi d'autres types de réseaux dont les MANET discontinus.

On qualifie aussi de communication opportuniste (*opportunistic networking*) le mode de communication s'appuyant sur le *store, carry and forward*, notamment lorsque les contacts entre nœuds ne peuvent pas être planifiés. Dans ce cas, l'accent est mis sur le fait que dans un MANET discontinu, les possibilités de communication peuvent être rares et qu'un terminal mobile saisit l'opportunité d'une rencontre, potentiellement brève, avec un ou plusieurs autres terminaux mobiles pour transmettre son message dans le but de le faire progresser vers sa destination.

Routage dans les MANET discontinus Des techniques très diverses ont été étudiées dans le cadre du DTN et de la communication opportuniste [152, 151, 129, 119]. En effet, une latitude importante est possible concernant plusieurs aspects tels que la politique de gestion des caches de messages ou le choix du ou des terminaux mobiles auxquels on transmet un message. Les caractéristiques du réseau (taille du réseau, schéma de mobilité des nœuds, temps de contact entre nœuds…) et le type de communication souhaitée (point à point, multi-points) influencent évidemment fortement la conception des protocoles de communication.

La problématique de communication est essentiellement vue sous l'angle du routage pour lequel on peut distinguer deux familles principales : le routage fondé sur le transfert par délégation et le routage épidémique.

3. `http://www.dtnrg.org`

Dans le transfert par délégation (*custody transfer*), défini dans [50], un terminal mobile stockant un message dans son cache tente de transférer ce message à un unique autre terminal mobile, lui déléguant du même coup la responsabilité du message. La délégation est faite lorsque le terminal rencontré est connu comme étant un meilleur porteur du message pour l'acheminer vers sa destination à travers une route spatio-temporelle. Le transfert par délégation convient bien aux situations dans lesquelles la mobilité des nœuds est prévisible voire contrôlée (cas ou les messages sont portés par des bus, des satellites, des robots…), c'est-à-dire aux situations dans lesquelles le choix du porteur suivant d'un message peut être fait de façon éclairée. L'approche est économique en ressources puisqu'une seule copie du message est présente dans le réseau, mais présente le risque de voire perdre le message si son unique porteur vient à défaillir. Par construction, le transfert par délégation est restreint aux communications point à point. Le *Bundle Protocol* [26, 136] est un exemple de protocole fondé sur le transfert par délégation.

Pour minimiser les risques de perte de message, la dissémination épidémique multiplie le nombre de copies du message. Un terminal mobile porteur du message en transfère une copie à chaque autre terminal mobile qu'il rencontre au gré de ses déplacements [64, 138]. Dans un processus similaire à celui de la propagation d'un virus, le message atteindra à terme tous les terminaux, et donc le terminal destinataire. Outre la robustesse vis-à-vis des défaillances de porteurs de messages, l'intérêt de cette approche est que le message atteindra sa destination avec une vitesse maximale. L'inconvénient majeur est la consommation de ressources importante tant du point de vue du trafic réseau que de l'espace de stockage des copies. Plusieurs techniques ont été proposées pour limiter le coût généralement considéré prohibitif de l'épidémie. Ceci est fait par exemple en arrêtant la propagation des copies rendues inutiles lorsque le message a atteint sa destination, par émission d'un message de contrôle jouant le rôle d'un antidote. On peut également limiter la portée des messages en limitant leur durée de vie ou la distance par rapport à l'émetteur lorsque l'on dispose de moyens de géolocalisation. Toute copie du message ayant dépassé la limite fixée est automatiquement détruit.

Des travaux récents sur la dissémination épidémique prônent une approche fondée sur l'exploitation de données contextuelles pour permettre à un terminal mobile de sélectionner parmi ses voisins les bons porteurs de messages. Le contexte pris en compte peut prendre plusieurs formes. Le plus souvent, l'hypothèse est faite que les terminaux mobiles sont portés par des humains et suivent donc des schémas de mobilité récurrents. Le contexte calculé par un terminal mobile comprend alors notamment un historique de ses contacts avec les autres terminaux, à la base de prédictions sur sa mobilité future, et donc sur son aptitude à porter utilement le message vers sa destination [97, 116, 94]. Des informations de nature sociale (données personnelles, appartenance à une communauté...) peuvent aussi être incluses dans le contexte [74, 120].

La dissémination épidémique est utilisée comme moyen de routage dans le cadre de communications point à point. Mais elle est évidemment aussi adaptée aux communications multi-points puisque qu'elle permet que plusieurs copies du message d'origine transitent dans le réseau et atteignent chacune un destinataire différent.

Communication basée destination et communication basée contenu Dans les MANET discontinus comme dans la plupart des réseaux, la communication est le plus souvent associée à un système d'adressage permettant à l'expéditeur de désigner explicitement le ou les destinataires des messages. À l'instar de ce qui est pratiqué dans le protocole IP, l'adresse de destination

embarquée dans le message sert à router le message au sein du réseau. On qualifie ce mode de communication basée destination (*destination-based communication*).

A contrario, dans la communication basée contenu (*content-based communication*), le message émis ne contient pas de destinataire explicite. C'est l'intérêt que portent les nœuds du réseau au contenu du message qui dicte son acheminement et sa réception. Les producteurs d'information injectent des messages dans le réseau par une opération de publication, et les consommateurs effectuent une souscription, déclarant leur intérêt pour certains messages. Le protocole d'acheminement se charge de faire en sorte que les consommateurs obtiennent les messages qu'ils souhaitent, en mettant potentiellement à contribution d'autres nœuds du réseau [36]. Ce mode de communication correspond à une des instances du paradigme de communication communément appelé *publish-subscribe* [48].

3.3 Approche orientée services

3.3.1 Concepts principaux de l'approche orientée services

L'utilisation du concept de service logiciel pour bâtir des applications, et notamment des applications distribuées, est maintenant largement répandue, et des domaines comme le SOA (*Service-Oriented Architecture*) ou le SOP (*Service-Oriented Programming*) sont en plein essor, le premier mettant l'accent plutôt sur l'utilisation des services comme briques de base pour architecturer une application, le second s'attachant à étudier les outils logiciels (essentiellement des intergiciels) permettant de programmer les applications à l'aide de services.

Notion de service L'approche orientée services structure une application autour de la notion de service. Un service est une unité fonctionnelle : il remplit une fonction identifiée contractuellement à travers un *descripteur de service* comprenant une interface fonctionnelle (description syntaxique). Le descripteur peut également contenir un ensemble de propriétés non fonctionnelles caractérisant le service ou des informations relatives au comportement du service.

Un service logiciel est aussi une unité de déploiement. Il doit pouvoir être déployé indépendamment des autres services. Après conditionnement, les codes mettant en œuvre le service, accompagnés d'une forme de description de leurs interfaces, sont généralement confiés à une infrastructure d'accueil qui se charge notamment de les rendre disponibles pour les autres services.

Le terme « service » est donc employé dans le sens d'un module logiciel jouant le rôle d'un *fournisseur* de fonctionnalité (ou fournisseur de « service », cette fois-ci au sens abstrait du terme). Ce module peut jouer le rôle de *client* de services lorsqu'il requiert d'utiliser d'autres services pour fonctionner.

Il faut noter que dans bon nombre de cas, les termes introduits ci-dessus ont une sémantique légèrement différente. On pourra par exemple considérer que le fournisseur est une entité logicielle délivrant potentiellement plusieurs services mis en œuvre par plusieurs objets de services (ou *servants*) implantant chacun l'interface d'un service. Le fournisseur applique éventuellement plusieurs politiques de gestion des objets de service (création à la demande, réserve pré-établie…).

Le terme fournisseur est même parfois compris dans un sens encore plus général, lorsqu'il désigne l'entité administrative hébergeant les objets de service.

Les caractéristiques énoncées plus haut sont très proches de celles que l'on attribue aux composants logiciels. En effet, les approches orientées composants et services ne se distinguent guère en ce qui concerne la définition des briques de base des applications, et les deux approches donnent des moyens similaires pour faciliter la réutilisation de code. C'est dans l'établissement des liaisons entre fournisseurs et clients que l'approche orientée services propose d'augmenter le découplage entre les entités formant l'application. La conception d'une application à base de services assemble en effet des services à partir de leurs descriptions. Les liaisons entre fournisseurs et clients, qualifiées de *liaisons tardives*, ne sont établies qu'à l'exécution, après une phase de *découverte* durant laquelle un client obtient, à partir d'éléments de description du service qu'il requiert, une référence vers un fournisseur de ce service.

Annonce et découverte de services La phase de découverte de services est une phase essentielle de l'approche orientée services. Un service a en effet la capacité d'établir à l'exécution une liaison avec d'autres services. Il doit donc disposer de moyens pour identifier puis choisir ces services, et obtenir une référence vers les services choisis. Un préalable à cette découverte est que les fournisseurs de services doivent annoncer les services qu'ils souhaitent rendre disponibles.

L'architecture la plus courante pour supporter l'annonce et la découverte de services repose sur la présence d'un annuaire de services (ou registre de services). Les fournisseurs de services enregistrent leurs services auprès de l'annuaire. Les clients de services interrogent l'annuaire en lui passant des informations décrivant les services recherchés. L'annuaire de services retourne alors un ensemble de descripteurs des services demandés avec, pour chacun d'entre eux, une liste de références à des fournisseurs de ces services. Le client peut appliquer une sélection supplémentaire en fonction des nouvelles informations acquises dans le but de se lier au meilleur fournisseur. La mise en œuvre de l'annuaire peut varier autant du point de vue des informations stockées que des protocoles d'accès par les fournisseurs et clients. Dans un contexte réparti, on peut mettre en place un annuaire centralisé, ou opter pour une mise en œuvre distribuée, avec duplication partielle ou totales des données.

Dans un certain nombre de cas, la mise en place d'un annuaire n'est pas souhaitable, par exemple lorsque l'on ne peut pas disposer d'une machine assez stable dans le réseau pour stocker un annuaire centralisé, ou parce que maintenir la cohérence d'un annuaire distribué serait trop coûteux. C'est notamment le cas dans certains réseaux mobiles ad hoc dans lesquels l'annonce et la découverte se font alors dans un mode pair à pair.

Invocation Une fois qu'un client de services s'est lié à un fournisseur de services, il peut invoquer les services fournis. Le mode principal d'invocation correspond au RPC (*Remote Procedure Call*) synchrone. Il s'agit pour le client d'invoquer une des méthodes (ou opérations) implantées par l'objet de service dont la référence a été donnée par le fournisseur et d'attendre une valeur de retour. Mais d'autres modalités d'invocation sont possibles (appel de méthode asynchrone, souscription/notification d'événements…).

Services avec état et services sans état Un service est dit avec état (*stateful*) si l'objet de service correspondant maintient un état tout au long de l'interaction qu'il a avec un client, cette

interaction faisant intervenir plusieurs invocations (l'état en question est un état dit conversationnel). Au contraire on parlera de service sans état (*stateless*) dans le cas ou la réponse à une invocation ne dépend pas du client ni des invocations passées.

3.3.2 Plates-formes à services

Plates-formes à services sur réseaux stables Plusieurs plates-formes intergicielles à services à vocation industrielle sont largement utilisées dans des applications distribuées, essentiellement sur des réseaux locaux ou sur Internet. Ces plates-formes implantent des spécifications couvrant tout ou partie des fonctionnalités attendues pour mettre en œuvre l'approche orientée services comme par exemple les courtiers de services Corba [122] ou Jini [6]. La plate-forme OSGi [126] n'héberge pas d'applications distribuées dans la mesure où tous les services sont encapsulés dans des *bundles* locaux mais elle permet le déploiement de bundles à partir d'entrepôts distants. Une extension nommée R-OSGi [133] a en outre été proposée pour rendre possible l'interaction de services distants.

Ce sont certainement les services Web qui ont le plus retenu l'attention des développeurs depuis quelques années. Le terme « services Web » couvre un ensemble assez vaste de technologies relevant de l'approche orientée services, les principales d'entre elles étant WSDL (*Web Service description Language* [148]) pour la description des services, SOAP [147] pour l'invocation et UDDI (*Universal Description, Discovery and Integration* [121]) pour la gestion d'annuaire distribué. Dans le cadre des services Web, il faut noter que le processus de découverte reste peu dynamique. UDDI sert surtout à l'annonce de services par et pour des entreprises, l'usage de l'annuaire de services étant essentiellement destiné aux ingénieurs (ingénieurs déployant les services côté fournisseurs et ingénieurs développant les applications clientes).

Découverte de services dans les réseaux dynamiques La mise en œuvre de l'approche orientée services a été étudiée également dans le cadre d'environnements moins stables, typiquement des réseaux locaux d'entreprise ou des réseaux domestiques dans lesquels des équipements peuvent être dynamiquement connectés. Il s'agit de réseaux filaires, de réseaux sans fil avec infrastructure (réseau Wi-Fi avec points d'accès par exemple) ou de réseaux sans fil ad hoc dans lesquels seules des communications à un saut sont possibles (réseau Bluetooth par exemple). Si les équipements peuvent être mobiles, la mobilité reste faible et les connexions et déconnexions d'équipements se font à un rythme lent. La problématique principale concerne la capacité à intégrer automatiquement les nouveaux équipements dans le réseau et à découvrir les services qu'ils fournissent. Des solutions ont été apportées à travers des produits et standards industriels comme UPnP [143], Bluetooth SDP [17] et SLP [55], et des propositions académiques comme DeapSpace [66], Ninja SDS [70] et INS [2] (une étude comparative de ces travaux est faite dans [153]). La plupart des efforts entrepris se sont focalisés sur le processus de découverte de services, en faisant l'hypothèse qu'une fois le fournisseur identifié par le client, l'invocation ne pose pas de problème particulier car elle peut s'appuyer sur des protocoles de transport standard comme TCP ou UDP.

Découverte de services dans les MANET Plus récemment, les caractéristiques propres aux MANET ont été prises en compte dans l'élaboration de protocoles de découverte de services

spécifiques. Il existe une grande variété de tels protocoles et leur classification n'est pas clairement établie malgré plusieurs études comparatives [146, 110, 73, 32]. De façon générale, l'objectif principal est de permettre un taux de découverte élevé (c'est-à-dire de faire en sorte que le maximum de clients souhaitant accéder à un service présent dans le réseau y parviennent) tout en réduisant le coût de cette découverte, ramené essentiellement à la quantité de trafic réseau engendrée. La rapidité de découverte est aussi un objectif, même s'il est quelque fois secondaire. Typiquement, on admet implicitement que la découverte doit se faire en un temps « raisonnable », variant selon les configurations de densité et de mobilité des nœuds du réseau, mais de l'ordre de quelques secondes au maximum.

Une première famille de protocoles, appelés protocoles de découverte de niveau réseau, couplent la découverte avec le routage (on parle aussi d'approche *cross-layer*). L'hypothèse de départ de ce type d'approche est qu'un algorithme de routage est de toute façon nécessaire à la découverte (et l'invocation) de services dans un MANET et que le protocole de routage transmet déjà des paquets permettant la découverte de nœuds. Il est donc judicieux que les protocoles de découverte de services exploitent ce trafic pré-existant. Les requêtes et réponses de découverte de services sont donc adjointes aux messages d'un algorithme de routage dynamique afin de diminuer la trafic réseau global. Cette approche peut être appliquée à des protocoles de routage existants (comme AODV dans [61] ou OLSR dans [79]) ou en construisant conjointement un algorithme de routage et de découverte [28, 47].

Une deuxième famille est composée des protocoles dits « de niveau applicatif », qui sont indépendants des protocoles de routage sous-jacents. Ces protocoles supposent généralement qu'un algorithme de routage point à point est disponible pour véhiculer certaines requêtes et réponses de découverte, bien qu'ils mettent aussi en œuvre le plus souvent une forme de diffusion (broadcast et multicast) fondée sur des transmissions à un saut. Les propositions de protocoles de niveau applicatif offrent un spectre de solutions très large. Les critères suivants permettent de les caractériser.

– *Architecture d'annuaire*
  L'architecture du système de découverte la plus simple est une architecture sans annuaire [65, 95, 29, 145]. On considère dans ce cas qu'il est difficile de trouver des nœuds disposant de suffisamment de ressources et offrant une accessibilité suffisante pour jouer le rôle d'annuaire. Chaque nœud conserve donc les descripteurs des services qu'il propose et mémorise éventuellement les descripteurs des services qu'il découvre dynamiquement dans le but de les rendre accessibles aux autres clients. La difficulté principale de cette approche réside dans le fait qu'a priori, il est nécessaire de diffuser périodiquement des annonces ou des requêtes de services dans tout le réseau (par inondation) pour permettre à un client de découvrir un service dont le descripteur est détenu par n'importe quel nœud, ce qui engendre un trafic réseau prohibitif. Des stratégies doivent donc être mises en place pour diminuer le trafic et éviter les transmissions redondantes.
  Une alternative est de maintenir un annuaire distribué sur un certain nombre de nœuds du réseau (il est exclu de maintenir un annuaire centralisé) [135, 118, 141, 83, 82]. Il peut s'agir d'un annuaire unique dupliqué sur tous les nœuds annuaires, ou la duplication des descripteurs peut n'être que partielle. Dans tous les cas, le maintien d'un contenu cohérent sur l'ensemble des nœuds annuaires engendre un trafic réseau significatif. Le placement des nœuds annuaires est généralement choisi dynamiquement, avec pour objectif de minimiser la distance (en termes de nombre de sauts) entre un client et le nœud annuaire le plus proche. Le plus souvent, le choix de mettre en place un annuaire est couplé avec la

présence d'un sous-réseau virtuel (*overlay network*) abritant les nœuds annuaires et servant de réseau d'acheminement des messages de découverte (annonces par les annuaires ou requêtes émanant des clients). On limite donc le trafic de découverte en le restreignant à ce sous-réseau. Ceci permet de maintenir un processus de découverte raisonnablement coûteux dans de « grands » réseaux (comportant plus d'une centaine de nœuds). En revanche, le maintien d'un sous-réseau structuré devient difficile en cas de forte mobilité des nœuds.

– *Mode de découverte*
  Deux modes de découverte sont utilisés. Dans la découverte pro-active (mode *pull*), un nœud intéressé par un service émet une requête de découverte (en broadcast, multicast, ou en unicast vers un nœud annuaire). Si un ou plusieurs nœuds peuvent satisfaire la requête, une ou plusieurs réponses seront renvoyées au demandeur. Dans le cas d'une découverte réactive (mode *push*), les nœuds apprennent l'existence des services au gré de leurs déplacements, en prenant connaissance des descripteurs diffusés dans leur voisinage direct ou indirect. Les annonces sont diffusées par les fournisseurs en mode broadcast à un saut ou en multicast (éventuellement limité à k sauts). Si la découverte repose sur la présence d'annuaires, le mode *push* n'est généralement pas implanté.

– *Description et sélectivité*
  Il existe un spectre assez large dans la complexité des descripteurs de services. L'approche la plus fréquente est de restreindre la taille des descripteurs pour minimiser à la fois la taille de l'espace consacré à leur stockage et le trafic dû à leur dissémination. Cependant, plusieurs travaux mettent en avant le fait qu'enrichir les descripteurs permet une meilleure sélection par les clients et accroît l'efficacité globale. Par exemple, un client capable de sélectionner le service le plus proche ou situé sur un nœud fournissant un bon temps de réponse permettra de réduire le trafic de façon globale en évitant des invocations infructueuses ou devant parcourir une grande partie du réseau [47]. Des langages de description (typiquement des dialectes XML) sont proposés pour incorporer aux descripteurs des informations de qualité de service, de contexte, de portée, etc.
  Il faut noter que la richesse des descripteurs est presque systématiquement destinée à affiner la sélection et n'est pas exploitée dans le protocole de découverte. Toutefois, certains travaux intègrent une partie de la phase de sélection dans le protocole de découverte en prenant en compte des informations supplémentaires comme une classification des services en groupes [29] voire des informations contextuelles comme la proximité géographique [108]. L'un des buts de cet enrichissement est d'être capable d'opérer une sélectivité dans l'acheminement des messages et du même coup de réduire le trafic réseau engendré par le protocole.

La quasi totalité des protocoles de niveau applicatif évoqués ci-dessus ont été conçus pour des MANET connexes et font usage d'un protocole de routage dynamique point à point ou multicast supposé disponible. Dans le processus de découverte proprement dit, quelques-uns d'entre eux, qui fonctionnent sans annuaire ni sous-réseau virtuel, font exception en se contentant de communications à un saut (en unicast et broadcast) pour mettre en œuvre une forme de dissémination [95, 108]. De ce fait ils sont susceptibles de supporter la fragmentation du MANET. Toutefois, dans tous les cas, le résultat de la découverte, obtenu par le client, est une adresse de fournisseur de service et aucun protocole n'est défini pour l'acheminement des messages d'invocation (requêtes et réponses), messages qui sont donc implicitement destinés à être acheminés à l'aide d'un algorithme de routage point à point.

3.4 Plate-forme à services pour MANET discontinus

3.4.1 Architecture générale

Depuis 2003, l'équipe Casa s'est intéressée aux réseaux ad hoc discontinus. Nous avons notamment mené des travaux dans le cadre du projet ANR Sarah (Services asynchrones sur réseaux mobiles ad hoc, 2006–2009). Notre activité s'est focalisée sur deux aspects : d'une part la communication par messages et d'autre part la fourniture de services.

Notre approche générale a consisté à bâtir un ensemble d'intergiciels facilitant le développement d'applications distribuées pour réseaux ad hoc discontinus. Il s'agit d'une démarche visant à fournir des outils utilisables et à ne pas s'arrêter au développement d'algorithmes et protocoles, même s'ils sont partiellement validés par des simulations. Un objectif constant a été de produire des intergiciels qui peuvent être diffusés, de fournir des démonstrateurs, et de bâtir des expérimentations sur le terrain.

Nous avons développé une plate-forme intergicielle, dédiée au développement d'applications orientées services pour MANET discontinus, en adoptant une architecture en couches. La première couche constitue un intergiciel de communication opportuniste, basée contenu, et tolérant les ruptures de connectivité. Au-dessus de cet intergiciel de communication, baptisé DoDWAN (*Document Dissemination on Wireless Ad hoc Networks*), on a bâti un autre intergiciel constituant une plate-forme à services, appelée DiSWAN (*Distributed Services on Wireless Ad hoc Networks*), offrant donc des outils de découverte, sélection et invocation de services. La figure 3.3 montre les principaux composants de l'ensemble qui a été construit. Deux raisons principales ont guidé notre choix de structuration en deux couches :

– L'approche orientée services n'est évidemment pas la seule approche pour programmer les applications visant les MANET discontinus. L'utilisation directe du passage de messages (*i.e.* l'utilisation d'un MOM) est sûrement une alternative viable. Un développement fondé sur l'utilisation d'abstractions de plus haut niveau comme les *tuples spaces* (dans l'esprit de Lime [115] ou TOTA [102] par exemple) ou d'autres formes de partage de données (comme celle prônée par XMiddle [104] ou Transhumance [127]) peut être aussi envisagé. L'application de partage de documents décrite dans [11] est un autre exemple d'application relevant d'une telle approche alternative. Le fait d'isoler une couche logicielle fournissant les fonctions d'un MOM permet de faciliter l'application de plusieurs paradigmes de programmation.

– La fourniture de fonctions de base pour la communication dans un MANET discontinu est un problème suffisamment complexe pour justifier l'existence d'un intergiciel propre. Une approche monolithique mêlant les problématiques de communication et de fourniture de services peut se révéler difficile à gérer.

Il est à noter toutefois que ces deux couches intergicielles ne sont pas complètement disjointes. En effet, leurs mises en œuvre incluent des mécanismes relevant de techniques dites de *cross-layering* : des informations issues de la couche service sont directement injectées dans les entêtes des messages manipulés dans la couche communication. Réciproquement, des informations relatives à la couche communication sont rendues accessibles à la couche service et influent sur les processus de découverte ou d'invocation.

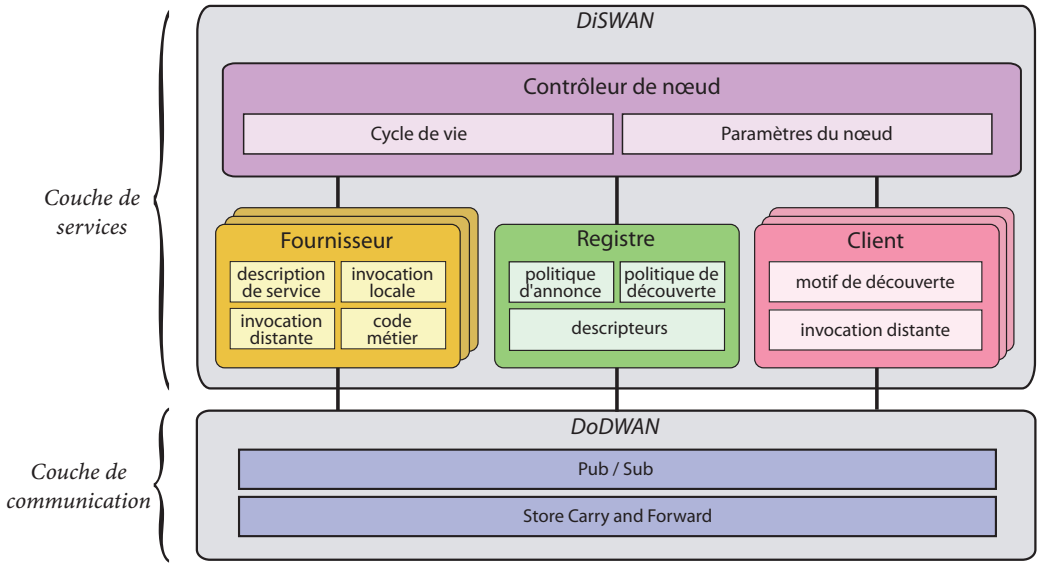


Figure 3.3 – Architecture générale de la plate-forme à services

3.4.2 Communication

L'intergiciel de communication supportant notre plate-forme à services met en œuvre un protocole de dissémination épidémique de documents basée contenu, en appliquant le principe du *store, carry and forward*. Sa définition complète et la description de son implantation, nommée DoDWAN, a fait l'objet de la thèse de Julien Haillot, effectuée sous la direction de Frédéric Guidec. Nous donnons ici un aperçu du protocole. Le lecteur intéressé en trouvera une description plus complète dans les articles [**i5**, 59].

Le protocole que nous avons conçu est un protocole épidémique reposant sur le principe du *gossiping* (bavardage). Les nœuds mobiles profitent de leurs contacts occasionnels pour s'échanger des informations, en fonction de leurs intérêts respectifs. La méthode exploitée dans notre protocole s'apparente à l'algorithme Autonomous Gossiping [39] dans la mesure où elle cherche à disséminer des messages dans tout le réseau et partage les objectifs suivants :

- éviter de bâtir et maintenir une infrastructure de routage des messages ;
- ne pas inonder le réseau, en évitant de transmettre des messages aux nœuds qui ne sont pas intéressés par ces messages ;
- minimiser les ressources (bande passante, mémoire) nécessaires à la dissémination des messages.

Le protocole fait usage de diffusion et de communication point-à-point. Dans le but de minimiser le nombre de message transmis, la diffusion est privilégiée par rapport aux transmissions point-point. Certaines technologies (comme Wi-Fi) permettent des communications en mode *unicast* fiables (en mettant en œuvre des mécanismes fondés sur des acquittements et de la ré-émission) alors que le mode *broadcast* reste non fiable, ce qui pourrait laisser à penser qu'il vaut mieux mettre en œuvre la diffusion par une séquence de transmissions point à point. Nous avons pourtant choisi de plutôt réduire les ressources mises en jeu, et donc d'éliminer le recours au transfert de trames d'acquittement ou aux ré-émissions, notre protocole étant intrinsèquement résistant aux pertes de messages : un nœud qui n'obtient pas un message de la part d'un de

ses voisins aura la possibilité de l'obtenir plus tard ou par un autre voisin qui en possède une copie. En outre, l'utilisation de la diffusion permet dans certains cas à des nœuds d'acquérir de l'information utile en recevant des messages qui ne leur étaient pas destinés a priori.

3.4.2.1 Entités du protocole

Afin de mettre en œuvre l'approche *store, carry and forward*, notre protocole met en jeu les entités décrites ci-dessous.

Documents, descripteurs et identifiants Un document est une unité de transfert composée de deux parties : un descripteur et un contenu. Le descripteur peut être vu comme une collection ouverte d'attributs servant à fournir tout type d'information sur le document, comme son origine, son sujet, une liste de mots-clés ou le type des données du contenu. Deux attributs doivent apparaître systématiquement dans le descripteur. Le premier est un identifiant unique qui sera utilisé notamment dans la phase de « bavardage » entre voisins et qui évitera la transmissions de doublons. Le deuxième attribut obligatoire consiste en une date d'expiration du document. Tout document dont la date d'expiration est atteinte est éliminé du réseau (on n'en conserve plus de copie), ce qui garantit que le nombre de messages présents dans le réseau ne croît pas indéfiniment.

De manière générale, on suppose que la taille d'un document est significativement plus grande que celle de son descripteur et que la taille du descripteur est elle-même significativement plus grande que celle de son identifiant. On aura par exemple des documents dont la taille est de l'ordre de 100 ko alors que les descripteurs ont une taille de l'ordre de 10 ko et que la taille d'un identifiant est de l'ordre de la dizaine d'octets. Notre protocole s'appuie sur ces contrastes pour minimiser le volume de données transmis. En effet, on échange le plus souvent possible de simples identifiants, afin de transférer uniquement les descripteurs utiles. Dans une même optique, on ne procède à un transfert de document que si un échange de descripteurs préalable à permis de s'assurer que ce transfert était nécessaire.

Du point de vue du protocole, les descripteurs de documents sont créés par le niveau applicatif. En pratique, il sont constitués à partir d'informations issues des applications (par exemple à partir de données utilisateur), d'une couche intergicielle supérieure (par exemple la couche service présentée plus loin dans ce chapitre), ou de données spécifiées par le propriétaire ou l'administrateur de l'équipement. En outre, les descripteurs ne sont pas immuables. Lorsqu'un document transite dans le réseau, son descripteur est susceptible d'être modifié par un nœud qui relaye le document.

La figure 3.4 montre un exemple de descripteur de document codé en XML. Celui-ci comporte les deux champs obligatoires prévus : un identifiant (sous la forme d'une chaîne de caractères générée automatiquement) et une date d'expiration (attribut *deadline*). Y figurent également des attributs stipulant l'applicatif l'ayant produit (*filesharing*), le type du document (PDF) et sa date d'émission ainsi qu'une liste de mots-clés. Un dernier attribut fait état d'un niveau de priorité haute, résultant par exemple d'un privilège accordé au nœud émetteur.

Cache Chaque nœud du réseau maintient un cache dans lequel sont stockés des documents. Notre protocole ne fait pas d'hypothèse sur la capacité du cache. Il est toutefois clair que cette

```
<descriptor
  id = "e3b5g76dt54e3z3"
  deadline = "Mon Feb 15 10:40:31 CEST 2010"
  date = "Mon Feb 15 08:55:22 CEST 2010"
  app = "filesharing"
  type = "PDF"
  keywords = "composant,service,intergiciel"
  priority = "high"
/>
```

Figure 3.4 – Descripteur de document

capacité est limitée, avec une taille propre à chacun des nœuds. L'élaboration d'une stratégie de gestion de cache ne fait pas partie des objectifs de nos travaux. Nous supposons qu'une politique quelconque est mise en place sur chacun des nœuds (nous avons par exemple implanté une politique FIFO). Il n'est en outre pas requis que la politique de gestion de cache soit unique sur l'ensemble des nœuds du réseau.

Profils d'intérêt Chaque nœud du réseau est sélectif vis-à-vis des documents qui lui sont adressés. Il possède en effet un profil d'intérêt caractérisant les documents qui l'intéressent et qu'il est donc disposé à recevoir, ou pour lesquels il est prêt à jouer le rôle de relais. Ce profil d'intérêt prend la forme d'un prédicat applicable aux descripteurs de documents. En appliquant ce prédicat au descripteur d'un document, un nœud décide s'il doit recevoir ce document et le placer dans son cache, ou bien l'ignorer. La figure 3.5 présente un exemple de profil d'intérêt. Les descripteurs correspondants doivent ici avoir été produits par l'applicatif *filesharing*, posséder une liste de mots-clés incluant « composant » ou « service », et être caractérisés par une priorité haute.

```
<profile
  app = "filesharing"
  keywords = ".*composant.*|.*service.*"
  priority = "high"
/>
```

Figure 3.5 – Exemple de profil d'intérêt

3.4.2.2 Description du protocole

Le protocole repose sur un modèle simple par lequel chaque nœud diffuse périodiquement une annonce informant ses voisins de son profil d'intérêt et des documents qu'il maintient dans son cache. Par ailleurs, chaque nœud réagit aux messages qu'il reçoit de ses voisins en émettant éventuellement des messages en réponse. Concrètement, un nœud effectue de façon itérative les trois phases décrites ci-dessous.

(a) Annonce de son catalogue et de son profil d'intérêt

Chaque nœud n_i diffuse périodiquement un message combinant une description de son propre profil d'intérêt $prof(n_i)$ et un catalogue $cat(n_i)$ composé des descripteurs des documents qu'il possède dans son cache et qui sont d'intérêt pour ses voisins. En diffusant son

propre profil d'intérêt, un nœud permet à ses voisins de découvrir quels sont les documents qui l'intéressent. Réciproquement, en recevant une telle information de la part de tous ses voisins, chaque nœud maintient une liste des profils d'intérêt de ses voisins *neighProf* (n_i) qui lui permet d'ajuster le contenu du catalogue qu'il diffuse périodiquement, évitant ainsi de transmettre un catalogue mentionnant des documents qui n'intéressent aucun de ses voisins.

Comme le voisinage de chaque nœud peut changer continûment, l'ensemble *neighProf* (n_i) est vidé juste après que n_i a diffusé son catalogue. Puis, à chaque fois que n_i reçoit un profil depuis un voisin, cette information est ajoutée à *neighProf* (n_i).

Tout nœud n_j qui reçoit un catalogue depuis l'un de ses voisins n_i examine les descripteurs qu'il contient pour identifier les documents dont les caractéristiques correspondent à son propre profil et qui ne sont pas encore dans son cache. S'il existe de tels documents, alors n_j construit une requête qui spécifie les identifiants des documents qu'il souhaite recevoir de n_i. Cette requête est ensuite envoyée en mode unicast à n_i. Une communication point à point est utilisée ici afin d'éviter que plusieurs voisins possédant le même document répondent de façon redondante à la requête.

(b) Traitement des requêtes

Après avoir envoyé son catalogue, le nœud n_i est susceptible de recevoir des requêtes de certains de ses voisins. Si n_i ne reçoit aucune requête, cela peut vouloir dire qu'il n'a à ce moment aucun voisin, ou qu'aucun de ses voisins n'est intéressé par les documents qu'il propose. Une autre raison peut être que le message d'annonce diffusé par n_i a été perdu ou que les requêtes qui en découlaient ont été perdues, à cause d'interférences radio transitoires. De telles erreurs de transmission ne sont pas critiques car un nœud mobile qui rate une opportunité d'échanger un document avec certains de ses voisins aura d'autres opportunités de le faire dans le futur, et éventuellement avec d'autres voisins.

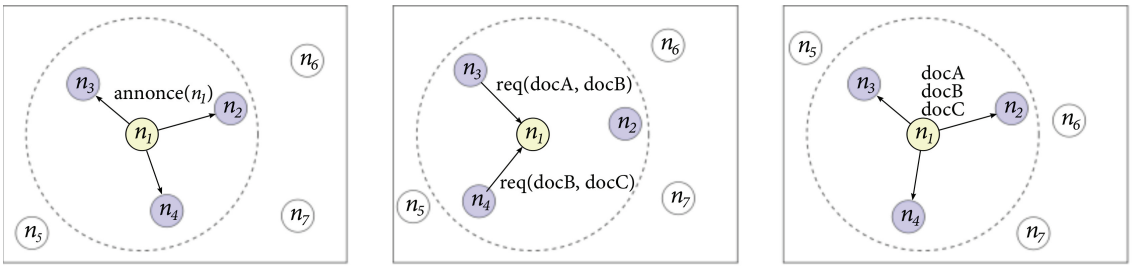
Si n_i reçoit des requêtes, ces requêtes sont traitées séquentiellement. Pour chaque identifiant présent dans la requête émanant d'un nœud n_j, n_i retrouve le document correspondant dans son cache et le diffuse. La diffusion est préférée à une transmission unicast même si la requête provient d'un nœud donné. En effet, n_i peut recevoir plusieurs requêtes pour le même document émanant chacune de voisins distincts. Il est plus efficace de satisfaire ces éventuelles requêtes redondantes en une seule diffusion. Pour éviter de diffuser plusieurs fois le même document, un nœud maintient la liste des documents qu'il a déjà diffusés depuis la dernière annonce de son catalogue, et ignore toute nouvelle requête d'un document figurant dans cette liste.

(c) Réception de documents

Quand un document est diffusé, le message peut être reçu par n'importe quel voisin. Tout nœud qui reçoit un document vérifie que le descripteur de ce document correspond à son propre profil d'intérêt, et que ce document n'est pas déjà dans son cache local. Si ces tests sont négatifs, le document est placé dans le cache. On peut noter que ce mode de transmission permet à un nœud de recevoir un document même s'il n'a pas émis de requête pour l'obtenir, mais uniquement parce qu'un autre nœud du voisinage en a demandé la diffusion. Cette situation peut résulter de la mobilité ou de la volatilité des nœuds, quand

par exemple un nœud vient de sortir d'un état de veille et capte la diffusion d'un document l'intéressant sans pour autant avoir eu l'occasion de recevoir préalablement le catalogue de l'émetteur du document. Les expériences que nous avons menées confirment que la possibilité pour un nœud d'obtenir des documents qu'il n'a pas explicitement demandés est une conséquence intéressante de notre décision de privilégier les diffusions par rapport aux communications point à point chaque fois que c'est possible : un nœud recevant un document « par hasard » s'abstiendra plus tard de demander explicitement ce document, ce qui contribuera à réduire le nombre et le volume des transmissions dans le réseau.

La figure 3.6 illustre l'enchaînement de ces trois phases dans un réseau de sept nœuds, en se focalisant sur un nœud.



(a) n_1 diffuse une annonce **(b)** n_1 reçoit des requêtes **(c)** n_1 diffuse les documents demandés

Figure 3.6 – Cycle des transmissions élémentaires autour de n_1

Optimisations Le protocole décrit ci-dessus a fait l'objet d'optimisations destinées à accélérer la dissémination des documents dans le réseau et à diminuer encore le volume de données transmises. Nous évoquons ici le principe des deux optimisations principales sans toutefois en donner le détail.

– *Relais immédiat au sein des îlots de communication*
Les réseaux ad hoc discontinus sont fragmentés en îlots de communication. Au sein d'un îlot, supposé connexe, il est possible de relayer immédiatement les messages en mettant en œuvre une forme de routage dynamique. Notre protocole inclut une forme de relais immédiat pour les messages diffusés ainsi que pour les messages transmis en mode unicast. Afin de limiter le coût d'une diffusion multi-sauts, un mécanisme inspiré d'OLSR est mis en place : chaque nœud définit périodiquement un sous-ensemble de ses voisins directs comme étant des relais multi-points (MPR : *Multi-Point Relays*). La diffusion s'effectuera uniquement via ces MPR, jusqu'à une certaine distance de l'émetteur, et la remontée des requêtes se fera en *source-routing*, également via ces MPR, vers l'émetteur de l'annonce. On étend ainsi la diffusion d'annonces et de documents au k-voisinage, accélérant du même coup la dissémination des documents dans le réseau.

– *Réduction du coût des annonces*
Dans le cas où le voisinage d'un nœud change lentement, la diffusion périodique d'annonces engendre un trafic inutile puisqu'une nouvelle annonce a de fortes chances de ne rien apprendre de nouveau aux voisins qui la reçoivent. Un mécanisme a été mis en place pour faire en sorte que les nœuds mémorisent les annonces reçues de leurs voisins. Si le contenu du catalogue proposé par un nœud (ou le contenu de son profil) n'a pas changé

depuis sa dernière annonce, il émettra un message d'annonce courte constitué simplement d'un identifiant faisant référence au contenu déjà diffusé. Une annonce longue, contenant l'ensemble du catalogue, ne sera émise que lorsqu'un changement le justifie. Ce changement peut être local, dû à la réception d'un nouveau document par exemple, ou peut découler d'un changement chez un voisin, dû par exemple à la modification de son profil d'intérêt. Dans le cas d'un voisinage durablement stable, le trafic se résume au bout d'un certain temps à un échange périodique peu coûteux d'annonces courtes.

3.4.2.3 Mise en œuvre

API Le protocole décrit plus haut a été implémenté en Java et forme l'intergiciel DoDWAN (*Document Dissemination in Wireless Ad hoc Networks*), implantant les services d'un MOM adapté à la communication basée contenu. DoDWAN présente une API fournissant les fonctionnalités principales suivantes :

- *Manipulation de documents*
  DoDWAN permet de créer des documents ayant un contenu quelconque. Un ensemble de fonctions autorise la manipulation de descripteurs attachés à ces documents (création, modification, traduction XML…).
- *Publication*
  Dans le paradigme de publication/souscription basée contenu, l'émission d'un document consiste en la simple publication d'un document. Ce document est injecté dans le réseau et c'est le contenu de ce document qui régira son acheminement vers sa ou ses destinations. Dans DoDWAN, une simple primitive publish prenant en paramètre un document est utilisée. Le document est alors déposé dans le cache et sa dissémination est automatiquement assurée, contrôlée par la présence d'attributs dans le descripteur attaché au document.
- *Souscription/réception*
  La réception de document nécessite une souscription préalable de la part de l'application. DoDWAN fournit une primitive subscribe prenant en paramètre un motif de document. Ce motif a la même forme qu'un descripteur mais autorise la spécification d'expressions régulières en lieu et place des valeurs d'attributs. La primitive subscribe prend un autre paramètre constituant une cible de document. Cette cible est un « processeur de documents » (objet disposant d'une méthode acceptant un document en paramètre) activé lors de la réception du document. Le profil d'intérêt associé à un nœud est en fait un ensemble de couples (motif, processeur de document). Ceci permet de différencier le traitement à effectuer lors de la réception effective du document, en fonction du motif de document qui a été employé pour décider de réceptionner le document. On peut ainsi faire cohabiter plusieurs applicatifs au-dessus de DoDWAN, qui pourront chacun (sans que ce soit forcément exclusif) prendre en charge un certain type de document reçu.

La figure 3.7 montre les codes exécutés respectivement par un nœud jouant le rôle d'émetteur et un nœud jouant le rôle de récepteur. L'émetteur constitue un document (objet de type Message) dont il renseigne le descripteur (durée de vie d'une minute et attribut « sujet » valant « DTN ») et indique la charge utile (ici une simple chaine de caractère), puis il publie son document. Le récepteur prépare quant à lui une souscription en définissant d'une part un processeur de document (objet de type Afficheur dont le rôle est d'afficher la charge utile du message – supposée être une chaîne de caractères – qu'on lui confie) et d'autre part un motif de souscription (qui fait

correspondre l'attribut « sujet » avec toute chaîne de caractères contenant « DTN » ou « SARAH »). Il dépose ensuite sa souscription en spécifiant un identifiant (ma_souscription) qui lui servira à l'annuler plus tard.

```
// Code de l'émetteur
Descriptor desc = new Descriptor();
desc.setDeadline(new Date(System.currentTimeMillis() + 60000));
desc.setAttribute("sujet", "DTN");
Message msg = new Message();
msg.setDescriptor(descriptor);
msg.setPayload(new String("Introduction aux MANET discontinus").getBytes());
dodwan.pubSubService.publish(msg);
```

```
// Code du souscripteur
public class Afficheur implements Processor<Message> {
   public void process(Message msg) {   // Affiche le contenu du message
      System.out.println(msg.getPayload());
   }
}
Descriptor motif = new Descriptor();
motif.setAttribute("sujet", ".*DTN.*|.*SARAH.*");
Subscriber souscr = new Subscriber(motif, new Afficheur());
dodwan.pubSubService.addSubscriber("ma_souscription", souscr);
...
dodwan.pubSubService.removeSubscriber("ma_souscription");
```

Figure 3.7 – Exemples de codes émetteur et souscripteur exploitant l'API DoDWAN

Caractéristiques principales La dissémination de documents dans DoDWAN repose sur des communications réalisées à travers une pile IP standard, les documents étant embarqués dans des datagrammes UDP. Ceci autorise l'exploitation de plusieurs technologies sous-jacentes. Les expérimentations on été menées essentiellement avec des interfaces Wi-Fi (IEEE 802.11), mais des tests ont également été effectués en utilisant Bluetooth (802.15.1) et sur des équipements PR4G (postes VHF utilisés par les forces armées européennes). La diffusion est mise en œuvre par le multicast UDP, dans un groupe arbitrairement choisi, et qui définit donc le réseau ad hoc cible.

Les documents manipulables par DoDWAN sont de taille quelconque. Afin de supporter le transport de longs documents, des mécanismes de fragmentation ont été mis en place. Un document ne pouvant pas être logé dans un seul datagramme est fragmenté en éléments qui sont eux mêmes des documents (partageant un même ensemble d'attributs dans leurs descripteurs). Ces fragments transitent à travers le réseau de façon indépendante. Lorsqu'un nœud finit par acquérir tous les fragments d'un même document, ces fragments sont localement réassemblés pour reconstituer le document d'origine.

DoDWAN place dans un cache les documents (complets ou fragmentés) qui ont été déposés localement par les applicatifs (via une publication) et ceux qui ont été reçus (après sélection) depuis

les nœuds voisins. Les documents sont maintenus dans le cache tant que leur date d'expiration n'est pas atteinte, sous réserve que le cache ne soit pas plein, et selon une politique FIFO : lorsqu'un document nouveau doit être placé dans le cache et que celui-ci est plein, les documents les plus anciennement placés sont détruits pour libérer de la place pour les nouveaux. Le cache est stocké en mémoire afin d'offrir de bonnes performances d'accès et est rendu persistant (sur le système de fichiers) pour résister à l'arrêt de l'intergiciel.

Couplage avec le simulateur MADHOC Les expérimentations de l'intergiciel DoDWAN que nous avons menées ont impliqué jusqu'à environ une vingtaine de nœuds. Afin d'étudier le comportement du protocole et d'évaluer les choix d'implémentation à une plus grande échelle, nous avons interfacé DoDWAN avec le simulateur de réseau MADHOC [72]. Ce simulateur a été développé par une équipe du Havre du LITIS, avec laquelle nous avons collaboré dans le cadre du projet ANR Sarah. Il présente l'avantage de pouvoir simuler des réseaux de grande taille, incluant potentiellement plusieurs milliers d'hôtes mobiles. MADHOC définit en outre un certains nombre de modèles de mobilité pré-définis, tels que le très classique modèle de Random Waypoint (dans lequel les mobiles évoluent en ligne droite et à vitesse constante vers une cible choisie au hasard, puis se fixent une autre cible à atteindre de la même façon), mais aussi des modèles plus « réalistes » de mobilité urbaine (circulation dans un centre commercial, dans les rues d'une agglomération, etc.). Dans le cadre du projet Sarah, nous avons interfacé les plates-formes DoDWAN et MADHOC, et avons en outre apporté de nouvelles fonctionnalités au simulateur MADHOC, telles que la possibilité de simuler la volatilité des hôtes mobiles.

3.4.3 Découverte de services

La découverte de services a pour objectif de permettre à un client de découvrir l'existence de services dans le réseau afin qu'ils puissent se lier à un fournisseur d'un de ces services. Les réseaux ad hoc discontinus posent un certain nombre de contraintes sur la découverte qui font que les solutions applicables aux réseaux ad hoc connexes, et a fortiori aux réseaux stables, ne sont pas réutilisables. On peut ramener ces contraintes à la difficulté pour un client d'établir une liaison stable avec un fournisseur de service. Le processus de découverte que nous avons mis en œuvre vise donc à permettre à un client de découvrir le maximum de fournisseurs proposant des services compatibles avec les besoins du client, tout en maintenant un coût de découverte raisonnable en termes de consommation de ressources.

3.4.3.1 Architecture de découverte

Dans les MANET discontinus encore plus que dans les MANET connexes, il est difficile de considérer qu'un quelconque nœud reste suffisamment accessible pour jouer le rôle d'annuaire. Les difficultés de communication, et notamment les délais induits par le mode *store, carry and forward*, rendent même impraticable une approche reposant sur un annuaire distribué sur un ensemble variable de nœuds. Nous avons donc logiquement choisi une architecture sans annuaire. Chaque nœud maintient un entrepôt local de descripteurs. Cet entrepôt stocke d'une part les descripteurs des services fournis par le nœud lui-même, et d'autre part les descripteurs des services qu'il a pu collecter dans le réseau. Chaque descripteur a une durée de vie, ce qui permet

de fonder une politique de gestion d'entrepôt de taille limitée : les descripteurs dont la durée de vie est dépassée peuvent être retirés de l'entrepôt.

C'est la couche de communication basée contenu, décrite plus haut dans ce chapitre, qui assure l'acheminement dans le réseau des descripteurs de services. Via une opération de publication, tout nouveau descripteur est disséminé dans l'ensemble du réseau, et les clients reçoivent les descripteurs qui les intéressent via une opération de souscription.

3.4.3.2 Descripteurs de services

Un fournisseur de service a la charge de construire un descripteur de service pour chaque service fourni, en utilisant toutes les informations qui seront utiles au client pour effectuer sa recherche. Un descripteur de service est composé de deux parties : un entête et un corps. L'entête contient un ensemble d'informations servant à la dissémination du descripteur dans le réseau. Il est destiné à être exploité par la couche de communication basée contenu, et donc lu par l'ensemble des nœuds participant à l'acheminement du descripteur. Le corps du descripteur de service consiste en un document WDSL détaillant essentiellement l'interface fonctionnelle du service. Ce corps n'est exploité que par la couche service lors de la sélection du service chez un client.

Concrètement, un descripteur de service est un document au sens de la couche de communication. L'entête de descripteur de service est formé par un ensemble d'attributs dont certains sont prédéfinis. Les principaux attributs prédéfinis sont les suivants : l'identifiant du descripteur, un type de document indiquant qu'il s'agit d'un descripteur de service, l'identifiant du fournisseur, le nom du service, sa catégorie, la date de création du descripteur et sa date d'expiration. La figure 3.8 montre un exemple de descripteur d'un service délivrant les horaires des marées du jour (pleine mer et basse mer) dans un port dont on fournit un identifiant.

La catégorie du service est définie relativement à une hiérarchie partagée par tous les nœuds du réseau et qui permet de grouper les services dans des catégories de plus en plus précises au fur et à mesure que l'on s'éloigne de la racine de la hiérarchie. Cet attribut permettra d'opérer un filtrage plus ou moins fin par les clients lors de leur recherche de service ou par les autres nœuds lorsqu'il s'agira de décider de relayer ou non les messages relatifs à un service donné. La hiérarchie est supposée fixe et cohérente sur l'ensemble des nœuds du réseau. Un exemple partiel d'une telle hiérarchie est donné dans la figure 3.9. L'implantation actuelle de la plate-forme DisWAN n'exploite qu'une seule hiérarchie permettant une catégorisation de nature fonctionnelle, mais étendre le système à plusieurs hiérarchies ne poserait pas de problème majeur. Cela permettrait un classement hiérarchique des services selon plusieurs critères (fournisseurs, prix…).

3.4.3.3 Processus de découverte

Un fournisseur de services effectue une publication des descripteurs des services qu'il propose. Cette publication se fait une seule fois. Il n'est en effet pas nécessaire de réitérer la publication car la couche communication se charge de maintenir en cache et de disséminer constamment les documents qui lui sont confiés. Il incombe toutefois au fournisseur de spécifier une durée de vie appropriée dans l'entête d'un descripteur de service afin que la dissémination de celui-ci cesse en temps voulu. La publication du descripteur de service se fait par un simple appel à la primitive publish de DoDWAN.

```
<descriptor
  id = "6hed819pa5c6fs90"
  deadline = "Mon Apr 12 12:22:41 CEST 2010"
  provider = "poseidon"
  type = "diswan-descriptor"
  createdAt = "Mon Apr 04 17:10:26 CEST 2010"
  serviceName= "horaireMaree"
  category = "service.tourisme.mer"
  sourceInfo = "SHOM"
  country = "France"
/>
```

```
<wsdl>
 <interface name = "horaireMaree">
  <operation name = "pleineMer">
   <input name = "port"  xs:type = "xsd:int"/>
   <output name = "result"  xs:type = "xsd:string"/>
  </operation>
  <operation name = "basseMer">
   <input name = "port"  xs:type = "xsd:int"/>
   <out name = "result"  xs:type = "xsd:string"/>
  </operation>
 </interface>
</wsdl>
```

Figure 3.8 – Exemple de descripteur de service

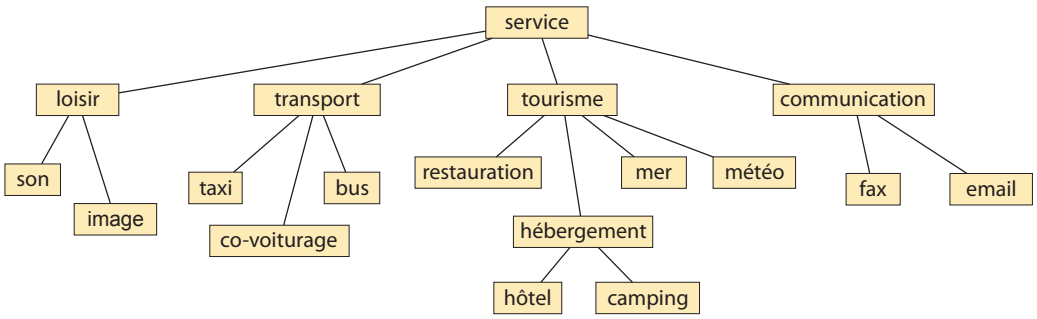


Figure 3.9 – Exemple de hiérarchie de services

Tout client de service doit effectuer dans un premier temps une souscription (via la primitive subscribe de DoDWAN) afin de collecter les descripteurs des services qu'il recherche. Cette souscription est construite avec un motif de service. Un tel motif est un entête de descripteur de service comportant des attributs similaires à ceux des descripteurs de service ordinaires, avec la possibilité d'employer des expressions régulières comme valeurs d'attribut. Le client a donc la possibilité de déclarer un intérêt pour une variété de services décrits de façon plus ou moins précise. Il peut par exemple inclure dans le motif le nom du service et le nom du fournisseur dans le cas où il souhaite exploiter un servant précis (en supposant qu'il connaisse ces détails à l'avance). Il peut au contraire effectuer une recherche beaucoup plus large en ne plaçant dans le motif de service qu'une catégorie de service. La dissémination des descripteurs de service se faisant sur l'ensemble du réseau, il est en général souhaitable que les clients fassent une recherche relativement ciblée afin de ne pas collecter un trop grand nombre de services. Des attributs non fonctionnels peuvent être utilisés à cette fin. La figure 3.10 montre un exemple de motif de souscription permettant de collecter tous les descripteurs de services portant le nom « horaireMaree » (dans les catégories relevant du tourisme ou de la météo), et dont la source d'information est le SHOM (Service Hydrographique et Océanographique de la Marine) ou l'UKHO (*UK Hydrographic Office*).

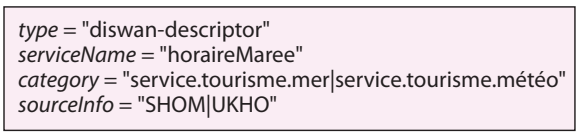


Figure 3.10 – Exemple de motif de souscription pour la collecte de services

Lors de la réception effective d'un descripteur, qui correspond à un motif spécifié dans l'une des souscriptions, celui-ci est placé dans l'entrepôt local de descripteurs de services. Une notification est émise pour renseigner le client sur cet événement. Le client peut opérer une sélection sur l'ensemble des descripteurs dont il dispose dans son entrepôt local, sélection qui peut porter non seulement sur les entêtes des descripteurs mais aussi sur les détails d'interfaces fonctionnelles présents dans les corps des descripteurs.

Il faut noter que l'entête de descripteur ne contient pas par défaut de description fonctionnelle du service. On considère en effet que cette description peut être complexe (incluant une hiérarchie de types par exemple) et, comme cela est fait pour les services Web, une définition est donnée *in extenso* en WSDL, dans le corps du descripteur. Dans le cas général, la souscription faite par les clients, et donc le contrôle de la dissémination, portent uniquement sur des aspects non fonctionnels et sur la catégorie de service. C'est seulement lors de la sélection dans l'entrepôt local qu'une recherche fonctionnelle est faite. Toutefois, il est possible de définir un attribut codant l'interface fonctionnelle du service et d'inclure cet attribut dans l'entête d'un descripteur, et donc aussi dans les motifs de services servant à la recherche de services par les clients. Dans l'exemple de la figure 3.8, un attribut « s:interface » pourra prendre la valeur "string pleineMer(int port) ; string basseMer(int port)". Cette façon de faire est tout à fait praticable dans le cas d'interfaces fonctionnelles relativement simples, à l'instar de ce qui est fait dans le cas des services Web dont l'invocation est possible via une URI.

3.4.4 Invocation de services

3.4.4.1 Principe

Une fois qu'un client a découvert un service, il doit être capable d'invoquer ce service. Plus précisément, l'objectif pour le client est d'envoyer une requête à un fournisseur du service, et de finalement recevoir une réponse calculée par ce fournisseur. Dans le cas des services sans état (qui ne requièrent pas de mémorisation de l'état conversationnel entre le client et le fournisseur), le client est la plupart du temps uniquement intéressé par le contenu de la réponse à sa requête, sans se préoccuper de savoir quel fournisseur a pris en compte sa requête.

Invocation basée contenu Dans notre plate-forme, une requête n'est a priori pas destinée à un fournisseur particulier mais spécifie seulement le service souhaité. On met donc en place un mécanisme d'invocation de service *basé contenu*. Le paradigme *publish/subscribe* est utilisé dans cette optique. Le client formule une requête qui inclut une version réduite du descripteur de service, et publie cette requête dans le réseau. Les fournisseurs souscrivent par défaut à tous les messages de requêtes portant sur les services qu'ils proposent. Ainsi, toute requête atteindra au final tous les fournisseurs « compatibles », c'est-à-dire susceptibles de servir la requête. L'objectif principal de cette approche est de maximiser les chances pour une requête d'atteindre rapidement un fournisseur capable de la servir, en dépit de la difficulté d'acheminement intrinsèque aux MANET discontinus.

La réponse du fournisseur est transférée en retour vers le client. Il s'agit d'un message à destination du seul client émetteur de la requête, bien qu'il soit acheminé en utilisant également le paradigme *publish/subscribe*. En effet, le descripteur de cette réponse contient un attribut « destination » dont la valeur est l'identité du client visé, et les clients posent par défaut une souscription pour les documents dont les descripteurs portent un attribut « destination » dont la valeur correspond à leur propre identité.

Une requête d'invocation est composée d'un entête incluant des attributs servant à désigner le service visé, et d'un corps consistant en un document SOAP décrivant l'appel de méthode proprement dit. De même, une réponse d'invocation comporte un entête faisant figurer notamment la destination de cette réponse. Le corps de la réponse est un document SOAP donnant la valeur de retour. La figure 3.11 donne un exemple de requête et de réponse associée.

Durée de vie Les invocations sont supposées tolérer les délais de communication induits par la nature même des MANET discontinus. La durée de vie d'une invocation est définie par le client à travers un attribut « deadline » qui est inclus dans la requête émise par le client mais aussi dans les réponses des fournisseurs. La couche de communication interprète cet attribut en arrêtant la propagation des messages concernés lorsque l'échéance spécifiée est atteinte. L'échéance est fixée en fonction de contraintes liées à l'application elle-même (la réponse doit par exemple être reçue avant une certaine date pour pouvoir garantir une forme d'interactivité) mais aussi en fonction de contraintes liées à la mobilité (le client sait par exemple qu'il quittera le réseau à une date donnée).

Politique de livraison des réponses Le fait que les requêtes d'invocation visent implicitement plusieurs fournisseurs potentiels plutôt qu'un fournisseur unique permet d'augmenter la

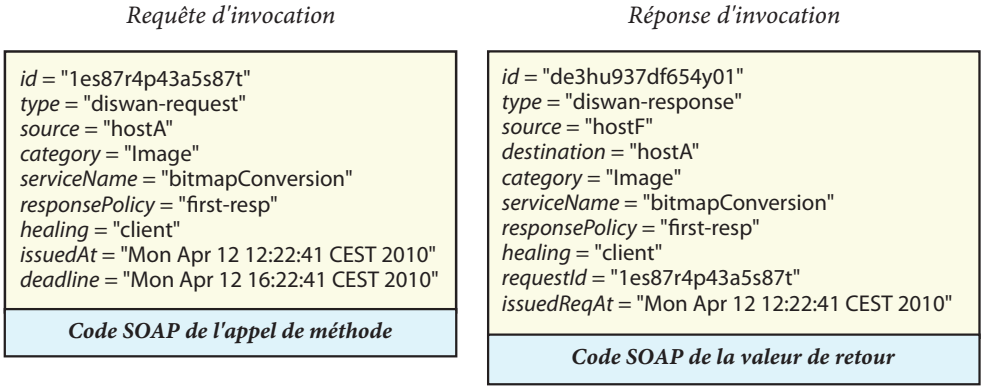


Figure 3.11 – Exemple de requête et de réponse d'invocation

réactivité du système puisque le fournisseur le plus proche pourra rapidement répondre. Toutefois, il génère aussi des réponses multiples pour une seule requête d'invocation. Nous avons proposé deux politiques de gestion de cette multiplicité des réponses.

Avec une première politique (nommée « multiple-resp »), toutes les réponses à une requête émise par un client sont délivrées à ce client. Il existe un certain nombre de situations dans lesquelles ce mode de livraison est intéressant pour le client. Considérons par exemple le cas d'un client invoquant un service de traduction de l'Anglais vers le Français. Des réponses multiples peuvent être utiles à l'utilisateur final qui pourra choisir la meilleure réponse. Dans le même esprit, il peut être intéressant dans un réseau de capteurs qu'une requête à un service de température fasse l'objet de la livraison de plusieurs réponses au client duquel émane la requête afin qu'il moyenne les valeurs reçues.

Au contraire, dans le cas où une seule réponse est suffisante, on adopte une politique « first-resp » avec laquelle seule la première réponse est délivrée au client alors que les suivantes sont ignorées. Par exemple, un service d'horaires de train autorisant des requêtes du type « à quelle heure ont lieu les départs pour Paris entre 12 h et 18 h ? » est normalement invoqué avec une politique « first-resp » dans la mesure où toute réponse reçue après la première est selon toute vraisemblance redondante.

Même si le plus souvent le choix de la politique de livraison est fait en fonction de la nature même du service (ou plus précisément de l'opération invoquée), le client spécifie la politique de son choix pour chaque invocation. Il peut donc s'il le souhaite adopter une démarche distincte pour les appels successifs à une même opération.

3.4.4.2 Réduction de la redondance des requêtes et réponses

Tirer parti du fait que plusieurs fournisseurs peuvent répondre à une requête n'est pas sans inconvénient dans le cas où une politique « first-resp » est appliquée, c'est-à-dire dans le cas où une seule réponse est effectivement exploitée par le client. En effet, les autres réponses sont ignorées quand elles arrivent au client mais les messages transportant les réponses continuent à se propager inutilement dans le réseau tant que leur durée de vie n'est pas atteinte, consommant ainsi de la bande passante et de l'espace de stockage dans les caches des nœuds intermédiaires. Nous avons mis en place des mécanismes pour réduire le surcoût induit par cette redondance en éliminant au maximum les messages de réponse mais aussi les messages de requête inutiles.

Il faut bien noter que les mécanismes habituellement employés pour résoudre ce genre de problème dans les réseaux connexes ne sont guère envisageables dans le cas qui nous intéresse car ils mettent en œuvre une forme d'agrément au sein de l'ensemble des fournisseurs afin de coordonner leurs réponses (en autorisant dans le meilleur des cas un seul fournisseur à répondre). L'absence de connectivité au sein des MANET discontinus proscrit ce genre d'approche, le processus d'agrément entre les fournisseurs (dont on ne connaît notamment pas la composition a priori) risquant fort d'être beaucoup trop lent à converger. L'objectif était donc non pas de garantir qu'une seule réponse parvienne au client, mais de tenter d'annuler autant que faire se peut la propagation des messages redondants.

Les trois techniques d'annulation suivantes ont été conçues : l'annulation réactive, l'annulation initiée par le client et l'annulation initiée par le fournisseur. Ces techniques appartiennent à la famille des techniques dites de « guérison du réseau » (*network healing*) employées dans le domaine de la dissémination épidémique (cf. paragraphe 3.2.2).

Annulation réactive L'annulation réactive est déclenchée par les fournisseurs sur réception de messages de réponses. Chaque fournisseur souscrit aux messages portant des réponses aux requêtes auxquelles il est lui-même susceptible de répondre, comme s'il était un client ayant invoqué un service qu'il propose. Cela permet aux fournisseurs de détecter les réponses redondantes. Quand un fournisseur F_1 reçoit une telle réponse émise par un fournisseur F_2, deux cas peuvent se présenter. Dans le premier cas, F_1 n'a pas déjà répondu lui-même à la requête correspondante. Il note alors qu'il doit s'abstenir de répondre, ceci jusqu'à la date d'expiration présente dans la réponse reçue. Dans le deuxième cas de figure, F_1 a déjà repondu à la requête ; il possède donc dans son cache de communication la réponse qu'il a faite (qui porte une date d'émission d_1) et la réponse qu'il vient de recevoir de F_2 (qui porte une date d'émission d_2). F_1 compare les dates d_1 et d_2 et annule la réponse la plus récente, c'est-à-dire qu'il arrête de participer à la dissémination de la réponse la plus récente en l'enlevant de son cache.

De plus, chaque fournisseur répondant à une requête annule également la requête reçue. Il est en effet inutile de contribuer à la propager. Quoi qu'il en soit, la requête circulerait à partir de ce fournisseur conjointement avec la réponse correspondante et le traitement décrit plus haut bloquerait la propagation de la réponse dès qu'elle serait reçue par un autre fournisseur.

L'annulation réactive est systématiquement appliquée car elle est toujours profitable du point de vue de la réduction du nombre de messages transmis (aucun message supplémentaire n'est introduit et certains messages ne sont pas transmis) et elle ne ralentit pas l'arrivée de la première réponse chez le client.

Annulation initiée par le client La deuxième technique d'annulation que nous avons employée est plus intuitive. Il s'agit de l'annulation initiée par le client. Quand un client reçoit une première réponse à sa requête, il démarre la dissémination d'un message de contrôle (de type *CureAll*) qui inclut l'identifiant de la requête et sa date d'expiration. Le rôle de ce message est d'informer les autres nœuds qu'ils doivent annuler toutes les requêtes et réponses correspondantes jusqu'à ce que cette date d'expiration soit atteinte. On suppose que les messages de contrôle circulent rapidement dans le réseau (en tout cas plus rapidement que les messages de requête et réponse), ce qui implique qu'un grand nombre de nœuds, si ce n'est l'ensemble des nœuds, doit avoir effectué la souscription nécessaire pour relayer ce type de message.

Ce mode d'annulation introduit des messages supplémentaires dans le réseau, ce qui tend à augmenter le trafic réseau global. Mais il faut noter que la taille des messages de contrôle est très petite. Par conséquent, le gain obtenu par l'annulation des réponses et requêtes redondantes (dont la taille peut être plus grande de plusieurs ordres de grandeur) compense dans la plupart des cas le surcoût dû aux messages de contrôle. Comme dans des conditions particulières – qui sont difficilement modélisables et qui n'ont pas été rencontrées en pratique dans nos expérimentations – cette technique d'annulation pourrait ne pas être profitable, elle n'est pas systématiquement appliquée. Le client l'active en positionnant un attribut spécial dans l'entête de sa requête.

Annulation initiée par le fournisseur La troisième technique d'annulation, appelée annulation initiée par le fournisseur, est une sorte de combinaison des deux précédentes. Elle consiste à faire démarrer la dissémination de messages d'annulation par les fournisseurs avant même que l'on sache qu'une réponse a été reçue par le client. Un fournisseur répondant à une requête d'invocation démarre la dissémination d'un message de contrôle, de type *CureOther*, similaire à un message *CureAll* mais qui inclut également l'identité F du fournisseur. Ce message sert d'ordre d'annulation de toutes les instances de la requête spécifiée dans le message de contrôle ainsi que de toutes les réponses correspondantes, à l'exception de la réponse émise par F. Un nœud du réseau ne réagit qu'au premier message *CureOther* pour une requête donnée, les suivants sont simplement ignorés. Un message de contrôle n'est utile que s'il est relayé par d'autres nœuds que les seuls fournisseurs du service concerné. On suppose encore une fois que ce type de message de contrôle est relayé par la quasi totalité des nœuds. L'objectif est qu'un message *CureOther* émis par un fournisseur F atteigne les autres fournisseurs plus vite que la réponse calculée par F, et par conséquent empêche d'avantage l'émission de messages redondants que ne le ferait l'annulation réactive.

Dans des conditions défavorables, il peut arriver qu'un message de type *CureOther* atteigne un fournisseur qui, dans le cas contraire, aurait été l'émetteur de la réponse qui aurait atteint le client en premier. En théorie, cette technique d'annulation peut donc retarder l'arrivée de la réponse chez le client. Les expérimentations qui ont été menées tendent à montrer que ce délai est négligeable alors que le gain en termes d'annulation de messages redondants est significatif. Intuitivement, on peut en effet comprendre qu'un mécanisme d'annulation de messages redondants est d'autant plus efficace que cette annulation est déclenchée tôt. Comme pour l'annulation initiée par le client, et dans la mesure où il existe un risque qu'elle ne soit pas rentable, l'annulation initiée par le fournisseur n'est activée que si le client la demande en insérant un attribut spécifique dans sa requête.

3.4.4.3 Invocation basée destination

Nous avons considéré dans les paragraphes qui précèdent qu'un client peut souhaiter invoquer un service indépendamment du fournisseur proposant ce service. Il existe toutefois des situations dans lesquelles un fournisseur spécifique est visé par le client lorsqu'il émet une requête d'invocation. Ceci est opportun parce que le fournisseur est considéré comme étant a priori le seul apte à assurer une certaine qualité de service, ou du fait de la nature même du service invoqué. Par exemple, si l'on considère un service délivrant des photos constamment renouvelées d'un point géographique donné, on peut penser que le service est implanté par un seul fournisseur localisé à cet endroit précis. Dans notre plate-forme à services DISWAN, ce cas de

figure est vu comme un cas particulier de l'approche générale basée contenu décrite plus haut. En effet, les attributs servant à caractériser une requête d'invocation peuvent contenir un attribut « destination » spécifié par le client afin d'identifier le fournisseur souhaité. Les fournisseurs ne répondent à ce genre de requête que si la valeur de cet attribut correspond à leur propre identité.

Dans le cas d'invocation basée destination, l'annulation initiée par le fournisseur est activée : dès que le fournisseur visé reçoit la requête, il initie la dissémination d'un message de contrôle qui annulera la propagation de cette requête qu'il est devenu inutile d'acheminer ailleurs.

3.4.4.4 Invocation de service avec état

Les services avec état requièrent que l'état de la conversation entre un client et un unique fournisseur soit maintenu pendant un certain laps de temps. Une séquence d'invocations a lieu pendant une *session*, dont l'état est généralement géré par le fournisseur. Dans le type d'environnement que nous avons visé, une session ne peut pas être considérée comme fiable, c'est-à-dire ayant lieu dans le cadre d'une liaison stable entre le client et le fournisseur. Dans un MANET discontinu, des ruptures de session sont probables, intervenant par exemple lorsque la machine hébergeant le fournisseur est mise en veille ou reste injoignable depuis le client pendant trop longtemps du fait de ses déplacements.

Nous avons étudié la faisabilité de mise en œuvre de sessions dans le cadre de l'invocation de service basée contenu et implanté une solution de rétablissement après rupture de session dans la plate-forme DISWAN. La démarche adoptée est inspirée de solutions proposées dans les MANET (comme les « *follow-me sessions* » décrites dans [63]) pour lesquelles la session est gérée par le client lui-même. L'information d'état de la session est retournée au client avec chaque réponse et est exploitée par le client pour qu'il soit en mesure de rétablir la session avec un autre fournisseur dans le cas où le fournisseur en cours n'est plus joignable. Le schéma d'invocation basée contenu permet d'effectuer une gestion de la rupture de session simple mais efficace.

Quand un client souhaite démarrer une session, il émet une requête d'invocation initiale portant un attribut « session ». Cette requête est disséminée dans le réseau, et potentiellement, plusieurs fournisseurs peuvent y répondre. Quand la première réponse, émise par le fournisseur F, atteint en retour le client, la session est considérée comme ouverte avec F du point de vue du client. Les invocations suivantes au sein de la session sont adressées à F, avec un acheminement basé destination. L'information d'état de session (typiquement un ensemble de valeurs de propriétés) maintenu par F est adjoint à chaque réponse. Tout fournisseur est supposé être capable d'exploiter ces informations pour rétablir à la demande un état conversationnel cohérent.

La session est considérée comme interrompue quand la date d'expiration de la dernière requête est atteinte. Le client rétablit la session en réémettant cette requête, cette fois-ci en mode basé contenu, c'est-a-dire sans attribut « destination », comme lors de l'émission de sa requête initiale. Cette requête contient les dernières informations d'état possédées par le client afin que la session puisse être rétablie avec un nouveau fournisseur compatible. Comme n'importe quel fournisseur compatible peut être utilisé pour le rétablissement de session, et étant donné que le rétablissement se fera avec le fournisseur le plus prompt à répondre, on peut considérer que le mécanisme est efficace du point de vue du temps de rétablissement.

3.4.5 Évaluation

La plate-forme à services DisWAN décrite dans les paragraphes précédents a été développée en Java, et des expériences ont été menées sur des réseaux formés de quelques nœuds mobiles. Nous avons par ailleurs exploité le couplage de l'intergiciel de communication DoDWAN avec le simulateur de réseau MADHOC (cf. paragraphe 3.4.2.3) dans le but d'observer le comportement de notre plate-forme dans des configurations plus variées, et notamment des environnements comportant un relativement grand nombre de nœuds. Je présente dans la suite de ce paragraphe quelques-uns des résultats obtenus.

3.4.5.1 Conditions de simulation

Environnement simulé Les simulations que nous avons menées portent sur un environnement réseau censé modéliser une zone urbaine avec des bâtiments et des contraintes de déplacements. Nous nous sommes efforcés de nous éloigner d'un comportement simpliste des nœuds du réseau (comme celui simulé par le modèle Random Waypoint) pour être le plus réaliste possible.

La zone étudiée est un rectangle de 500 m sur 900 m dans lequel une centaine d'individus peuvent se déplacer à l'intérieur de quatre bâtiments, et circuler en marchant d'un bâtiment à l'autre le long de routes. La vitesse des individus est variable, entre 2 et 5 km/h. À tout moment, un individu peut s'arrêter pendant un laps de temps d'une durée aléatoire comprise entre 1 et 6 minutes. Les nœuds du réseau représentent des utilisateurs équipés d'ordinateurs portables dotés d'interfaces radio ayant une portée de 20 m à l'intérieur d'un bâtiment et 50 m à l'extérieur. Chaque utilisateur a des affinités avec un des bâtiments (il a tendance à y rester plus longtemps que dans les autres) et ne peut pas accéder à la totalité des bâtiments de la zone pendant la durée de la simulation.

La plate-forme à services est déployée sur chacun des nœuds du réseau. Certains nœuds jouent le rôle de fournisseurs de services et d'autres le rôle de clients cherchant à utiliser l'un des services proposés. Les autres nœuds ne sont ni clients ni fournisseurs, mais sont utilisés comme relais de messages dans le processus de dissémination épidémique mis en jeu par la couche de communication. On définit le facteur d comme étant le facteur de duplication d'un service. Autrement dit, chaque service est déployé en d exemplaires, chacun sur un fournisseur différent. Dans les scénarios étudiés, les clients découvrent le service une seule fois. Ils effectuent ensuite une succession d'invocations tout au long de la période de simulation.

Mesures effectuées Deux types de mesures ont été effectués. Le premier objectif est d'estimer la satisfaction du client en termes de délai : pour la découverte, on mesure le temps écoulé entre le moment où le client déclare son intérêt pour le service par une opération de souscription et le moment où il obtient le descripteur de ce service. En ce qui concerne l'invocation, on mesure le temps entre le moment où le client envoie sa requête d'invocation et le moment où il reçoit la première réponse.

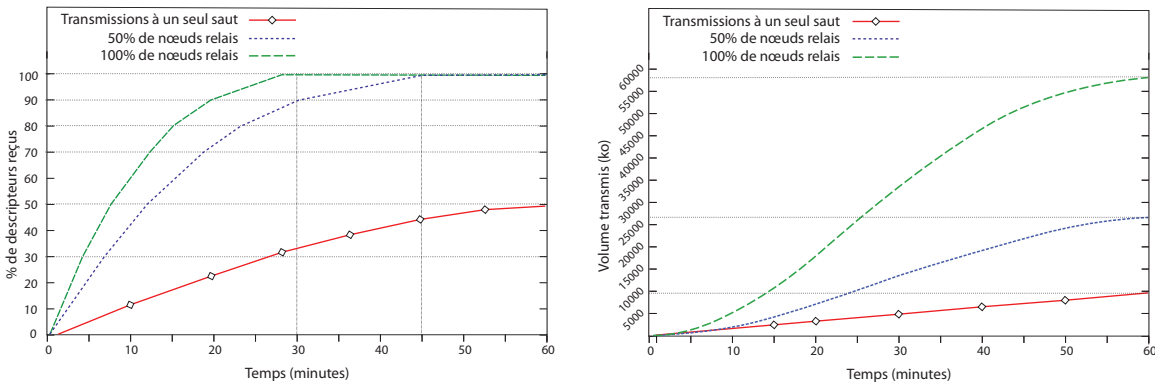
Le deuxième objectif est d'évaluer l'impact des protocoles de découverte et d'invocation sur la charge du réseau. On mesure pour cela le nombre total de messages émis et reçus ainsi que le volume global des données transmises.

3.4.5.2 Évaluation de la découverte

La performance du processus de découverte est directement liée à celle de la dissémination épidémique mise en œuvre par la couche de communication, la couche service n'apportant pas de mécanisme supplémentaire ayant un impact sur la dissémination. Des détails d'évaluation du protocole de dissémination peuvent être trouvés dans [59]. Nous montrons ici les résultats concernant la découverte dans l'environnement décrit plus haut, avec la configuration suivante : le réseau est formé de 80 nœuds parmi lesquels 8 jouent le rôle de fournisseurs et 32 jouent le rôle de clients. Les 40 autres nœuds sont potentiellement relais pour la dissémination de messages. Les fournisseurs proposent globalement 4 services différents, déployés chacun sur 2 fournisseurs différents (on a donc $d = 2$).

La figure 3.12-(a) montre l'impact du nombre de relais participant à la dissémination sur la satisfaction des clients. Elle indique le pourcentage cumulé de descripteurs reçus en fonction du temps dans trois cas de figures différents :

- lorsque la moitié des nœuds participent à la dissémination des messages ;
- lorsque tous les nœuds participent à la dissémination des messages ;
- lorsque la dissémination est désactivée. Cette troisième courbe sert de courbe de référence : le module de *store, carry and forward* a été désactivé sur tous les nœuds, ce qui impose que les clients doivent passer à proximité des fournisseurs pour recevoir leurs descripteurs, via une communication à un saut. On mime ainsi un processus de découverte plus classique dans les MANET (similaire à celui d'UPnP par exemple). Les descripteurs sont diffusés périodiquement pour pallier l'absence de mécanisme de dissémination.



(a) satisfaction des clients

(b) charge réseau

Figure 3.12 – Performance de la découverte

On peut voir sur la figure que l'utilisation de la dissémination épidémique sélective mise en œuvre par la couche de communication apporte un gain significatif de vitesse de découverte. Après 30 mn, 90 % des clients ont découvert le service qu'ils recherchent quand la moitié des nœuds relayent les messages. Ce pourcentage atteint 100 % quand tous les nœuds sont prêts à jouer le rôle de relais. Après le même laps de temps, seuls 30 % des clients sont satisfaits quand la communication est limitée par des transmissions à un saut.

Effectuer une dissémination épidémique pour publier les descripteurs produit potentiellement plus de messages en comparaison avec l'utilisation de diffusion à un saut. Cependant, le protocole

mis en œuvre dans la couche de communication a été conçu pour minimiser le nombre et la taille des messages échangés, afin que le surcoût soit le plus faible possible. Nous avons évalué la charge réseau induite par le processus de découverte dans les trois cas décrits plus haut. La figure 3.12-(b) montre les courbes obtenues. La charge réseau est définie comme le volume cumulé émis par tous les nœuds. Les résultats de mesure confirment que la charge croît linéairement dans le cas de transmissions à un saut, en raison de la diffusion périodique des descripteurs de services. La dissémination épidémique entraîne quant à elle une charge qui croît plus vite, mais cette charge tend à se stabiliser car le protocole de bavardage sous-jacent reste très frugal lorsque les nœuds ont tous pris connaissance des descripteurs.

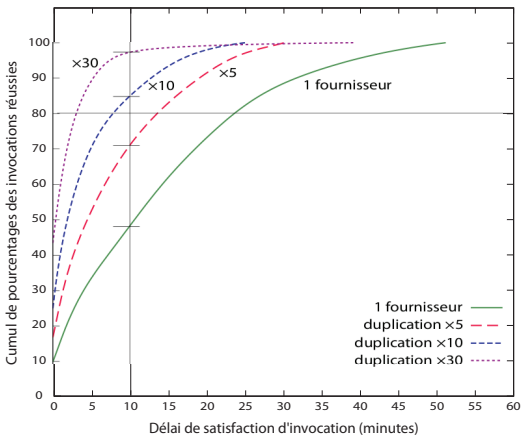
3.4.5.3 Évaluation de l'invocation

Multiplicité des fournisseurs L'approche basée contenu est censée accélérer le processus d'invocation quand plusieurs fournisseurs proposent le même service. Nous avons mesuré l'impact du facteur de duplication de fournisseurs (le facteur d cité plus haut) sur le délai de satisfaction des requêtes d'invocation dans le cadre de l'invocation basée contenue implantée dans la couche service de notre plate-forme. Nous avons comparé cet impact à celui obtenu dans un cadre plus classique d'invocation basée destination, c'est-à-dire quand les requêtes d'invocation sont envoyées à un fournisseur précis, préalablement découvert. Dans le scénario étudié, 100 nœuds sont présents dans le réseau. Un seul service est fourni par un nombre variable de fournisseurs, selon le facteur de duplication choisi. Un client unique cherche à effectuer une succession d'invocations à ce service, en étant intéressé par la première réponse. La totalité des nœuds participent à la dissémination des messages.

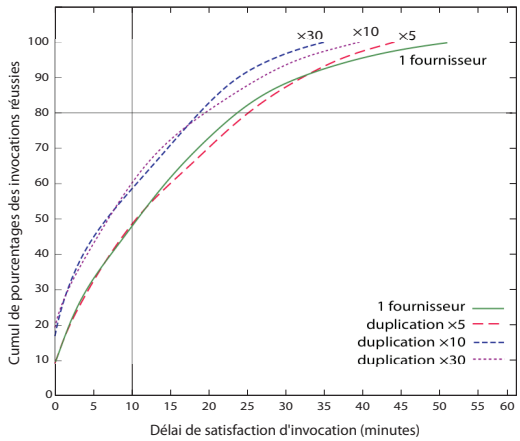
La figure 3.13 illustre la comparaison effectuée. Chaque courbe donne le pourcentage cumulé d'invocations réussies en fonction du temps. Par exemple, sur la figure 3.13-(a), on voit que 70 % des invocations aboutissent en moins de 10 minutes quand le facteur de duplication est de 5, c'est-à-dire quand le service est fourni par 5 des 100 nœuds du réseau.

Globalement, on voit que l'utilisation d'une approche basée contenu (figure 3.13-(a)) permet de tirer parti de la multiplicité des fournisseurs d'un même service : l'augmentation du facteur de duplication des fournisseurs accélère de façon importante les invocations. Il est en outre intéressant de constater que le gain est significatif même pour des facteurs de duplication relativement faibles. En revanche, la duplication des fournisseurs a peu d'effet quand les invocations sont basées destination (figure 3.13-(b)).

Mécanismes d'annulation Nous montrons ici les résultats d'évaluation des mécanismes mis en place dans notre plate-forme à services pour éliminer autant que possible les messages redondants induits par le fait que de multiples fournisseurs peuvent répondre à la requête d'un client intéressé seulement par la première réponse. Les résultats obtenus portent d'abord sur la réduction du nombre de messages transmis lors du processus d'invocation. La figure 3.14-(a) montre le cumul du nombre de messages de réponses émis par les fournisseurs en fonction du temps lorsque l'on applique les trois techniques d'annulation décrites au paragraphe 3.4.4.2. Une courbe de référence a été ajoutée qui donne les résultats obtenus sans annulation. Le gain est substantiel, quelle que soit la technique d'annulation appliquée (jusqu'à près de 5 fois moins de messages transmis à terme lorsque l'annulation réactive et l'annulation initiée par le fournisseur sont employées).



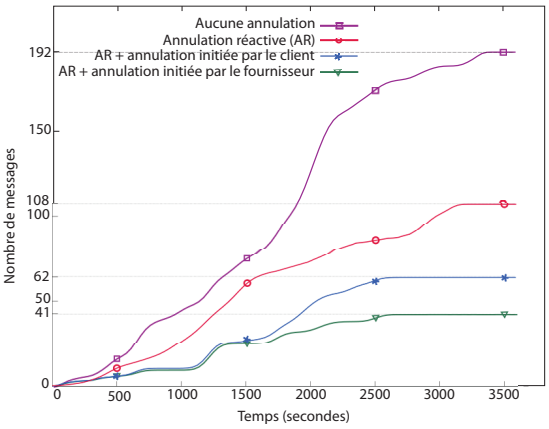
(a) Invocations basées contenu



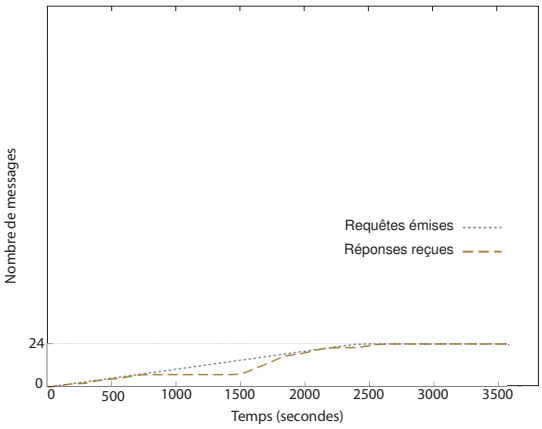
(b) Invocations basées destination

Figure 3.13 – Impact de la multiplicité des fournisseurs sur la satisfaction d'invocation

Même si l'annulation initiée par le fournisseur peut en théorie retarder les messages de réponse, voire empêcher ceux-ci d'arriver au client, ceci ne se vérifie pas en pratique pour le scénario considéré : le retard reste faible et aucune réponse n'est perdue. Ceci est illustré par les deux courbes de la figure 3.14-(b) qui montrent l'évolution du nombre cumulé de requêtes envoyées par le client et du nombre cumulé de réponses effectivement reçues par ce client (en activant l'annulation réactive et l'annulation initiée par le fournisseur). Les réponses sont, comme on pouvait le prévoir, légèrement retardées mais 100% d'entre elles sont effectivement reçues au bout au bout de 2600 secondes.



(a) Annulations de réponses

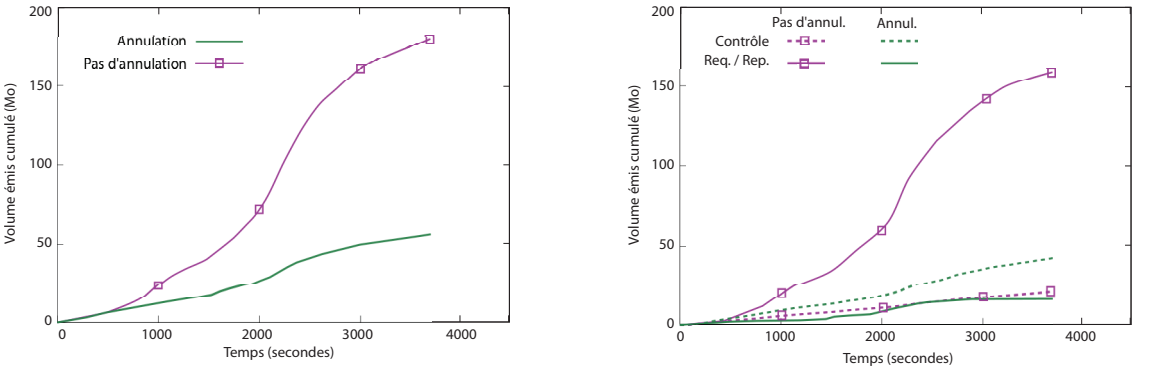


(b) Correspondance requêtes/réponses

Figure 3.14 – Effet des mécanismes d'annulation sur le nombre de messages transmis

L'impact de l'annulation est encore plus manifeste lorsque l'on considère le volume de messages transmis durant le processus d'invocation. Des messages supplémentaires sont introduits dans le réseau mais ceux-ci sont de petite taille et permettent d'éviter la dissémination de messages qui peuvent être relativement long. En pratique, le volume de requêtes et réponses économisées compense largement le surcoût des messages de contrôle. Ceci est confirmé par les résultats donnés dans la figure 3.15-(a). Cette figure montre la charge réseau, cumulée dans le temps, induite

par les messages émis par tous les nœuds du réseau lorsque qu'aucune technique d'annulation n'est appliquée et quand les trois techniques d'annulation décrites au paragraphe 3.4.4.2 sont conjointement utilisées. La charge réseau est divisée en deux parties : d'une part le volume global des messages « utiles », c'est-à-dire des requêtes et réponses proprement dites (la taille de chaque message de requête et réponse est de 26 ko), et d'autre part, le volume des messages de contrôle, incluant notamment les messages ajoutés par les techniques d'annulation. Comme illustré dans la figure 3.15-(b) on constate que le volume de messages de contrôle double (il augmente de 27 Mo) quand l'annulation est activée, mais ce surcoût est très petit en comparaison des 142 Mo requis en plus pour les requêtes et réponses quand l'annulation n'est pas activée. Au total, l'annulation permet de passer d'un volume émis de 182 Mo à 60 Mo, soit un gain d'un facteur supérieur à 3.



(a) Vue globale

(b) Détail : requêtes/réponses & messages de contrôle

Figure 3.15 – Volume transmis cumulé

3.4.6 Bilan

Les travaux décrits dans ce chapitre ont été menés pour une bonne part dans le cadre du projet ANR Sarah (Services Asynchrones pour Réseaux Mobiles Ad Hoc [4]) . Ce projet a été financé par l'ANR pour la période 2006–2009 dans le cadre l'appel à projets ARA SIIA (Action de recherche Amont « Sécurité, Systèmes embarqués et Intelligence Ambiante »). Outre l'équipe Casa du Valoria, ce projet a fait intervenir des équipes du LaBRI (Bordeaux), du XLIM (Limoges) et du LITIS (Le Havre). La contribution de Casa a porté principalement sur les aspects relatifs à la communication et à la fourniture de services dans les MANET discontinus, les deux autres thèmes du projet portaient d'une part sur la sécurité et d'autre part sur la modélisation et la simulation des algorithmes. Nos contributions dans les domaines couverts par Casa ont fait l'objet de plusieurs publications. Je suis co-auteur des articles suivants :

- Nicolas Le Sommer, Salma Ben Sassi, Frédéric Guidec, Yves Mahéo. A Middleware Support for Location-Based Service Discovery and Invocation in Disconnected MANETs. *Studia Informatica Universalis*, 8(3):71–97, septembre 2010 [**j1**].
- Yves Mahéo, Romeo Said. Service Invocation over Content-Based Communication in Disconnected Mobile Ad Hoc Networks. *In Proceedings of the 24th International Conference on Advanced Information Networking and Applications (AINA'10)*, Perth, WA, Australie, avril 2010, pages 503–510, IEEE CS [**i1**].

4. `http://www-valoria.univ-ubs.fr/SARAH`

- Nicolas Le Sommer, Romeo Said, Yves Mahéo. A Proxy-based Model for Service Provision in Opportunistic Networks. *In Proceedings of the 6th International Workshop on Middleware for Pervasive and Ad-Hoc Computing, 9th International Middleware Conference (Middleware'08)*, Louvain, Belgique, décembre 2008, IEEE CS [**i2**].
- Romeo Said, Yves Mahéo. Toward a Platform for Service Discovery and Invocation in Disconnected Mobile Ad Hoc Networks. *In Proceedings of the 5th International Conference on Embedded and Ubiquitous Computing (EUC'08)*, Shanghai, Chine, décembre 2008, pages 238–244, IEEE CS [**i4**].
- Yves Mahéo, Romeo Said, Frédéric Guidec. Middleware Support for Delay-Tolerant Service Provision in Disconnected Mobile Ad Hoc Networks. *In Proceedings of the Workshop on Java and Components for Parallelism, Distribution and Concurrency, International Parallel and Distributed Processing Symposium (IPDPS'08)*, Miami, FL, USA, avril 2008, IEEE CS [**i3**].
- Frédéric Guidec, Yves Mahéo. Opportunistic Content-Based Dissemination in Disconnected Mobile Ad Hoc Networks. *In Proceedings of the International Conference on Mobile Ubiquitous Computing, Systems, Services and Technologies (UBICOMM'07)*, Papeete, Tahiti, novembre 2007, pages 49–54, IEEE CS [**i5**].

Intergiciel de communication

Les MANET discontinus sont des réseaux dynamiques considérés comme particulièrement « difficiles » dans lesquels la communication est un problème en soi, essentiellement du fait de l'absence de connectivité de bout en bout. Nous avons proposé un protocole de communication pour MANET discontinus relevant du DTN et de la communication opportuniste. Notre approche a été de nous appuyer sur un modèle de communication épidémique basée contenu dont la mise en œuvre réduit autant que possible le volume des échanges afin d'offrir un bon compromis entre la rapidité d'acheminement des messages et la consommation de ressources. L'une des caractéristiques majeures de ce protocole est qu'il ne fait pas d'hypothèse particulière sur les schémas de mobilité des nœuds du réseau même si, bien évidemment, son efficacité dépend de facteurs généraux comme la densité des nœuds et leur taux de mobilité. Notre approche contraste donc avec celle appliquée dans de nombreux travaux actuels, tels que ceux du projet européen Haggle [5] qui cherchent à exploiter des schémas de mobilité – typiquement des schémas de mobilité sociale – supposés récurrents pour guider l'acheminement des messages.

L'intergiciel DoDWAN implante le protocole que nous avons conçu. Il offre une API très simple, de type *publish/subscribe* qui permet la dissémination de documents au sein d'un MANET discontinu. DoDWAN est distribué en open-source [6]. Cet intergiciel a été conçu de manière à pouvoir mettre en œuvre un ensemble de protocoles variés, et pas seulement un protocole fondé sur la dissémination épidémique basée sur le contenu. Nous devrions donc nous appuyer à l'avenir sur DoDWAN pour étudier d'autres protocoles de communication, notamment des protocoles exploitant la délégation (*custody transfer*) pour mettre en œuvre des communications point-à-point efficaces.

5. http://www.haggleproject.org
6. http://www-valoria.univ-ubs.fr/CASA/DoDWAN

Intergiciel de fourniture de service

Nos travaux sur la fourniture de services ont été de nature plus exploratoire et difficilement comparables à d'autres propositions concurrentes. À ma connaissance, il n'existe en effet à ce jour pas de résultats de travaux de recherche sur l'approche orientée services étudiée spécifiquement dans le cadre de MANET discontinus (hormis les travaux décrits dans [128] qui se focalisent sur l'évaluation analytique d'un support de services simple fondé sur des communications à un saut). Cette thématique constitue pourtant une thématique clé du domaine émergeant de l'« *opportunistic computing* » [35], comme le suggère le démarrage, en septembre 2010, du projet européen SCAMPI (*Service platform for social-aware mobile and pervasive computing*) [7]. Ce projet a pour ambition d'apporter des réponses aux problèmes de la fourniture de services dans les réseaux opportunistes, en exploitant notamment les relations sociales entre utilisateurs.

Nous avons pour notre part proposé une plate-forme supportant la découverte et l'invocation de services qui tire parti d'une couche de communication de publication/souscription basée contenu. La découverte de services a été directement implémentée à l'aide de mécanismes de publication/souscription basés contenu. La difficulté à mettre en contact les fournisseurs de services et les clients potentiels de ces services est le problème central de la fourniture de services dans ce type de réseau. Il faut donc être capable d'exploiter au maximum les possibilités de relations entre clients et fournisseurs. Le premier atout de notre approche est que la recherche se fait sur tout le réseau sans payer le prix d'une inondation rédhibitoire. La deuxième piste que nous avons suivie est d'enrichir la description de services et la formulation des souhaits des clients de façon à élargir le plus possible leur mise en correspondance. Dans cette optique, nous avons proposé des descripteurs de services ouverts, autorisant une recherche non limitée aux aspects fonctionnels des services. Pour permettre un appariement client-fournisseur plus large, on pourrait envisager d'employer des techniques relevant des services Web sémantiques [81]. Dans le cas de clients suffisamment évolués, la recherche de services, et donc indirectement de fournisseurs, pourrait se faire en fonction de critères sémantiques, moins sélectifs que des critères purement syntaxiques, afin d'avoir accès à une offre plus grande de services malgré les difficultés de communication.

L'invocation de service pose des difficultés particulières dans les MANET discontinus. En effet, si dans des environnements connectés, il est tout a fait possible que les clients contactent un fournisseur précis postérieurement à une phase de découverte de plusieurs fournisseurs, les difficultés de communication inhérentes aux MANET discontinus forcent les clients à être capables de contacter tout fournisseur du service qu'ils demandent. Nous avons développé des techniques d'invocations de service fondées sur la publication/souscription basée contenu afin de permettre une exploitation maximale de la présence potentielle de plusieurs fournisseurs aptes à servir une requête d'invocation donnée. Plusieurs techniques de guérison du réseau ont été implantées pour atténuer l'augmentation de trafic engendrée.

Simulations

Un certain nombre d'expérimentations de simulation ont été effectuées pour évaluer les protocoles que nous avons développés, qu'il s'agisse des protocoles de communication implantés dans DoDWAN ou des protocoles de découverte et d'invocation de services de DiSWAN. Les

7. http://www.ict-scampi.eu

simulations se sont appuyées sur un interfaçage de l'intergiciel DoDWAN avec le simulateur de réseau MADHOC, développé dans le cadre du projet ANR Sarah. En employant ce simulateur, nous avons pu faire tourner le code quasi inchangé de notre intergiciel (seul le réseau était simulé, l'intergiciel étant plutôt émulé), ce qui a permis de mesurer l'impact non seulement des algorithmes mis en en jeu mais aussi de tenir compte des choix d'implantation effectués.

Il faut souligner que l'évaluation par simulation des intergiciels pour MANET est une tâche difficile. La simulation exploite un certain nombre de modèles plus ou moins conformes à la réalité : il s'agit de modéliser tout d'abord le réseau ciblé en définissant un modèle de l'environnement physique, de la propagation radio et de la mobilité des nœuds. Mais il faut aussi modéliser l'application exploitant l'intergiciel, y compris éventuellement le comportement de l'utilisateur de l'application, afin de définir des scénarios complets. Un certain nombre d'approximations et de biais peuvent rendre les résultats de simulation peu exploitables (comme souligné dans [86], l'état de la crédibilité des résultats de simulation publiés dans la littérature n'est pas fameux). Dans le cadre des recherches sur les MANET discontinus, la difficulté est accrue du fait de l'impact souvent drastique des paramètres temporels liés à la mobilité, ce qui rend la comparaison des approches proposées rarement possible. Il faut sans doute être très prudent sur l'interprétation des résultats de simulation, qui permettent, à mon sens, de dégager seulement des ordres de grandeur et non pas de comparer finement des algorithmes.

Dans l'optique de réduire les biais liés à l'élaboration des scénarios de simulation, on peut citer un effort récemment entrepris pour collecter des traces d'expérimentation sur des réseaux sans fil, dans le cadre du projet CRAWDAD (*Community Resource for Archiving Wireless Data At Dartmouth* [8]). L'un des objectifs est de disposer d'une base commune de traces de mobilité réelles pouvant servir de base à des *benchmarks* pour les applications et intergiciels. Cette base est régulièrement augmentée mais elle couvre un large spectre de technologies et de scénarios de communications (avec et sans infrastructure) et est pour le moment trop incomplète pour pouvoir être exploitée de façon systématique.

Expérimentations sur le terrain

Afin de compléter les simulations, nous avons mis en place une série de campagnes d'expérimentation portant sur l'exécution d'applications réelles s'exécutant sur des MANET discontinus. Plusieurs applications classiques relevant de la communication entre utilisateurs ont été développés au dessus de l'intergiciel DoDWAN : une application de courrier électronique (MailWAN), de forums de discussion (NewsWAN), de messagerie instantanée (ChatWAN), de partage de fichiers (FishWAN). Une dernière application (Présence) typique des réseaux mobiles a été ajoutée : elle permet de visualiser la liste des machines présentes dans le voisinage radio à tout instant.

Ces applications ont été déployées sur mini-PC (*netbooks* Dell Inspiron mini 10) qui ont été distribués à des promotions d'étudiants, à l'ENS Cachan (antenne de Bruz) et à l'UBS (Vannes). Les étudiants disposaient à leur guise de leurs mini-PC pendant la période d'expérimentation. Plusieurs campagnes ont été menées, portant sur des promotions allant de 6 à 25 étudiants, et s'étalant chacune sur une période allant d'une dizaine de jours à une année universitaire entière.

Les mini-PC disposaient d'une interface Wi-Fi configurée en permanence dans le mode ad hoc. Les utilisateurs n'avaient pas la possibilité d'exploiter cette interface radio en mode infrastructure

8. http://crawdad.cs.dartmouth.edu

en se connectant aux points d'accès pourtant présents dans leur environnement. Bien entendu, les mini-PC étaient régulièrement mis en veille et éteints par les utilisateurs. L'ensemble des mini-PC formait donc un réseau mobile ad hoc discontinu, souvent peu dense.

Sur chaque mini-PC, l'activité de l'intergiciel DoDWAN et des applications de communication était enregistrée en permanence. Des fichiers de traces étaient ainsi constitués afin de rendre compte des échanges effectués entre les machines. Ces fichiers de traces ont été collectés de façon régulière, pendant l'expérimentation. Pour cela, l'utilisateur était invité à utiliser ponctuellement l'interface Ethernet filaire pour relier sa machine à Internet afin de rapatrier les fichiers générés sur une machine de collecte gérée par notre équipe. Cette collecte continue n'avait pas de caractère obligatoire, les machines disposant d'assez d'espace de stockage pour conserver les traces durant toute la durée de l'expérimentation. Mais l'objectif était d'une part de se prémunir contre des pannes de machines qui auraient entraîné la perte des traces et d'autre part de nous permettre de disposer de données avant la fin de l'expérimentation, notamment dans le but de détecter et corriger au plus tôt des erreurs d'implantation du protocole de communication ou des applications.

À ce jour, l'exploitation des traces est toujours en cours. Elle nécessite la définition d'un ensemble de métriques spécifiques au protocole de DoDWAN et le développement d'outils d'analyse appropriés. Les premières estimations confirment que notre protocole semble bien remplir son rôle dans des conditions variées, mais il est trop tôt pour dresser un bilan des expérimentations.

Du côté applicatif, quelques enseignements commencent aussi à être tirés. On peut citer l'exemple de l'application de partage de fichiers FishWAN. Cette application propose des services à l'utilisateur semblables à ceux que l'ont peut trouver dans des applications de partage de fichiers en mode pair à pair sur Internet : mise à disposition des fichiers (publication), requêtes pour exprimer des souhaits d'obtention de fichiers (souscription), réception asynchrone des fichiers demandés, etc. Une différence notable réside dans les modalités d'obtention des fichiers qui, dans FishWAN, sont fondées uniquement sur le contenu. Un utilisateur de FishWAN doit souscrire à des fichiers en construisant des filtres de réception. Ces filtres sont des expressions régulières sur des attributs (couples clé/valeur) de descripteurs. Il est à la charge des utilisateurs de remplir le mieux possible les descripteurs des fichiers qu'il publient afin que les autres utilisateurs puissent fabriquer des filtres adéquats pour les recevoir. Le procédé s'est révélé trop souple pour être facilement exploitable. Les utilisateurs ne partagent en effet pas forcément un ensemble de mots clés a priori et ont des difficultés à définir leurs filtres de réception et leurs descripteurs de publication. De toute façon, on s'aperçoit qu'ils sont peu enclins à se contraindre à manipuler explicitement des ensembles d'attributs. Il apparaît donc nécessaire de compléter FishWAN par un dispositif de découverte de fichiers (comme celui que nous avons mis en œuvre dans la plate-forme à services DiSWAN) qui se contenterait d'une configuration minimale de la part de l'utilisateur.

De façon générale, on constate la nécessaire adaptation du comportement des utilisateurs au caractère intrinsèquement asynchrone des communications. En effet, les performances des réseaux d'infrastructure donnent une impression d'instantanéité qui ne se retrouve pas dans un MANET discontinu. Il convient de tenir compte de cet état de fait pour définir les fonctions offertes à l'utilisateur et leur présentation.

4
Conclusion

Ce mémoire a décrit les travaux de recherche que j'ai menés dans l'équipe Casa du Valoria depuis 2002. Trois projets, intitulés Concerto, Cubik et Sarah ont constitué le cadre de ces recherches. Ces trois projets ont eu pour objet commun la construction d'intergiciels pour le développement d'applications distribuées ciblant des réseaux dynamiques. Ils se différencient d'abord dans les caractéristiques du dynamisme des réseaux considérés. Dans le projet Concerto, les applications étudiées étaient des applications relevant du Grid Computing et visaient des grappes de stations de travail banalisées. Le dynamisme de ces réseaux restait toutefois relativement modéré. Il était essentiellement induit par la fluctuation des ressources offertes par chacune des machines du réseau. Dans le projet Cubik, c'est un environnement d'informatique ambiante qui était considéré, impliquant un dynamisme plus important : un ensemble de machines hétérogènes, potentiellement volatiles et mobiles constituaient la cible de composants logiciels. Cependant, la mobilité des machines était faible et leur nombre était borné. C'est avec le projet Sarah que nous avons ciblé des réseaux encore plus dynamiques, en proposant un intergiciel de communication et une plate-forme à services pour réseaux mobiles ad hoc discontinus. Ce dernier type de réseau est toujours considéré comme très difficile (on parle de *challenged networks* dans la littérature). Dans ce dernier cas en effet le réseau visé est un ensemble ouvert d'équipements mobiles et volatiles, et on ne fait pas d'hypothèse de connectivité de bout en bout.

Les trois projets décrits dans ce mémoire se différencient aussi par les paradigmes mis en avant pour bâtir les applications distribuées. Prenant acte du fait que les applications distribuées pour réseaux dynamiques étaient des applications de plus en plus complexes, nous avons pris en compte des avancées du génie logiciel qui proposent de fonder l'architecture des applications sur les concepts de composants et de services logiciels. Le projet Concerto a permis la définition d'une forme de composants dits parallèles, qui encapsule un code parallèle et distribué dans un composant. Dans le projet Cubik, nous avons proposé un modèle hiérarchique de composants ubiquitaires fondé sur Fractal, permettant à un composant de fournir ses fonctionnalités sur l'ensemble du réseau dynamique, malgré les ruptures de connectivité. Les aspects relatifs à l'architecture des composants et à leur déploiement autonome ont été étudiés. Dans le projet

Sarah, c'est l'application de l'approche orientée service qui a été explorée. Nous avons proposé une plate-forme intergicielle facilitant la découverte et l'invocation de services dans des réseaux mobiles ad hoc discontinus.

L'ensemble des travaux menés a permis de dégager les principales contributions suivantes :

Projet Concerto

- Un modèle simple de composant parallèle permettant d'encapsuler un code parallèle dans un composant logiciel.
- Un intergiciel de découverte et d'observation de ressources présentes dans une grappe de stations de travail banalisées, les ressources considérées incluant des ressources système, des entités de l'architecture de composants et des ressources logicielles définies par l'application.

Projet Cubik

- Un modèle de composants hiérarchique, issu de Fractal et adapté aux environnements ubiquitaires.
- Une méthode de déploiement originale, fondée sur la prise en compte de contraintes de déploiement portant à la fois sur l'environnement cible (en termes de ressources) et sur l'architecture de l'application telle qu'elle a été spécifiée par le concepteur.
- Un intergiciel permettant le déploiement de composants ubiquitaires et leur exécution distribuée.

Projet Sarah

- Un protocole de communication dédié aux réseaux mobiles ad hoc discontinus. Ce protocole est fondé sur le principe du *store, carry and forward*, et de la communication opportuniste basée contenu.
- L'intergiciel DoDWAN mettant en œuvre ce protocole, accessible à travers une API de type publication/souscription.
- La plate-forme à services DiSWAN, bâtie au-dessus du protocole DoDWAN, autorisant le développement d'applications orientées services. Les fonctions de découverte mais aussi d'invocation ont été adaptées au contexte particulier des réseaux mobiles ad hoc discontinus.

Il existe à l'heure actuelle peu de résultats théoriques sur lesquels on puisse s'appuyer pour concevoir des algorithmes pour systèmes distribués dynamiques. La formalisation même des caractéristiques de tels systèmes n'est pas encore établie [10], même si quelques travaux tentent de contribuer à la conception d'outils formels adaptés (comme par exemple les graphes évolutifs [51, 24]). Nos derniers travaux ont porté sur des réseaux très dynamiques comme les MANET discontinus, qui forment sans doute une des instances les plus difficiles de systèmes distribués du fait de l'absence d'hypothèse sur la connexité du réseau et même sur la stabilité de composantes connexes au sein du réseau. Du côté applicatif, on entrevoit des possibilités d'exploiter les réseaux dynamiques pour fournir de nouveaux services visant notamment les utilisateurs nomades. Mais les besoins sont encore mal cernés, et on les exprime souvent à l'identique des besoins en vigueur dans le monde connecté, avec la contrainte de mobilité en plus. Or des applications entièrement nouvelles, n'ayant de sens que dans un contexte de mobilité, devraient sans doute pouvoir apparaître.

Ce contexte nous conforte dans la démarche que nous avons progressivement mise en place. Il s'agit d'une démarche que l'on pourrait qualifier d'exploratoire, qui privilégie le développement de prototypes d'intergiciels et leur utilisation expérimentale. Les concepts, algorithmes et techniques que nous avons conçus ont en effet systématiquement fait l'objet de développements de logiciels, parfois lourds (effort de développement de l'ordre de 20 h.m. pour l'intergiciel DoDWAN par exemple), afin d'en vérifier la faisabilité. Peu à peu, nous nous sommes orientés vers une validation combinant la simulation et l'expérimentation sur le terrain. Cette approche, qui participe d'une vision expérimentale de la recherche en informatique, me semble très complémentaire de celle fondée sur la mise en œuvre d'algorithmes pour lesquels des propriétés – notamment de correction – sont formellement établies, dans le but de rendre les réseaux très dynamiques réellement exploitables.

En suivant cette démarche, on peut prolonger nos travaux récents sur les MANET discontinus. Plusieurs perspectives sont ouvertes, que ce soit sur les aspects protocolaires ou sur des aspects liés aux intergiciels de plus haut niveau comme les intergiciels à services. Je cite pour clore ce mémoire quelques pistes de recherche, dont certaines sont déjà suivies dans le cadre de travaux de thèses menés dans notre équipe.

Géolocalisation

Une voie pour améliorer la communication et la fourniture de services dans les MANET discontinus est d'exploiter des informations de géolocalisation des équipements du réseau. De plus en plus, les terminaux mobiles sont équipés de dispositifs permettant leur positionnement géographique, typiquement grâce à un récepteur GPS. Ces informations de positionnement peuvent être à la base d'optimisations de l'acheminement des messages dans le réseau. Dans un modèle de communication épidémique basée destination, on peut par exemple restreindre l'ensemble des voisins à qui on procure une copie des messages en tenant compte de la position de ces voisins et de la position du nœud destinataire, ceci afin de limiter le nombre de copies du message et d'atteindre plus vite le destinataire. Nous avons commencé à explorer cette approche dans le cadre de la fourniture de services (thèse de Salma Ben Sassi, encadrée par Nicolas Le Sommer et Frédéric Guidec). Le développement d'une plate-forme à services spécifique [**j1**] nous a permis d'étudier la possibilité d'exploiter la géolocalisation, non seulement au niveau de l'acheminement des messages, mais aussi dans la mise en œuvre des phases de découverte et d'invocation. Dans cette plate-forme, un fournisseur de service a la possibilité de définir, pour chacun des services qu'il fournit, une zone de découverte (resp. une zone d'invocation), limitant ainsi la possibilité pour les clients de découvrir (resp. d'invoquer) un service. Au cours de l'acheminement des messages d'annonce et d'invocation, la comparaison de la position d'un nœud intermédiaire avec les zones définies par le fournisseur permettent de décider si ce nœud doit relayer les messages.

La géolocalisation ne se limite pas forcément à la fourniture de coordonnées géographiques. Dans un cas général, et notamment quand on ne dispose pas d'équipement comme un récepteur GPS, il est souhaitable de pouvoir décrire une position de manière abstraite et parfois de manière relative. On voudra par exemple repérer un terminal mobile comme étant « au 1er étage du bâtiment sud ». Fournir un modèle de positionnement à la fois suffisamment général et exploitable dans un MANET discontinu, par essence dépourvu de moyen de partager facilement des connaissances communes, reste un problème difficile. La plate-forme citée plus haut jette

les bases d'un modèle de positionnement original permettant notamment la manipulation de positions décrites de manière abstraite et la définition de zones hiérarchiques.

Réseaux hybrides

Les travaux que nous avons menés jusqu'ici dans l'équipe Casa concernaient les MANET discontinus dans lesquels tous les terminaux sont mobiles. Nous nous sommes placés d'emblée dans un contexte difficile, dans lequel il n'est en effet pas possible de s'appuyer sur des équipements plus stables que d'autres. Les résultats que nous avons obtenus montrent que la communication et la fourniture de services peuvent être mises en œuvre, même si une qualité de service proche de celle que l'on peut attendre des techniques visant des MANET connexes ne peut pas être atteinte. On peut dans une certaine mesure estimer que ce type de configuration de réseau reste relativement marginal, et considérer les domaines applicatifs visés comme des niches (communication de crise, partage d'information sur le terrain, etc). Dans beaucoup de cas en effet, les équipements mobiles évoluent plutôt dans un environnement où sont aussi présentes des machines stables voire fixes. C'est typiquement le cas dans un contexte urbain. Il est donc légitime de penser à exploiter les ressources de ces machines fixes, que ce soit en termes de capacité de calcul et de stockage qu'en termes de connectivité.

Une nouvelle piste de recherche pour notre équipe est d'étudier la fourniture de services dans des réseaux que l'on peut qualifier d'hybrides dans la mesure où ils sont formés d'équipements mobiles mais aussi d'infostations. Ces infostations sont fixes et peuvent constituer des passerelles entre un ou plusieurs équipements mobiles et le réseau d'infrastructure fixe. On ne fait toutefois pas l'hypothèse que ces infostations sont en nombre suffisant ni que leur déploiement est planifié pour former une infrastructure couvrant un territoire donné. On a donc affaire à un réseau hybride discontinu : il est toujours nécessaire de mettre en œuvre des techniques relevant de la communication opportuniste et du paradigme *store, carry and forward* pour permettre aux terminaux mobiles de communiquer. La présence d'infostations doit cependant permettre de faciliter l'acheminement des messages, sous réserve que de nouveaux protocoles soient mis au point pour exploiter le caractère hybride du réseau. Le travail d'Ali Makke, qui a débuté sa thèse sous ma direction en octobre 2010 devrait permettre d'aboutir à des propositions concrètes sur ce sujet.

Sécurité

Le développement d'une intergiciel pour MANET discontinus ne peut pas être complet sans une prise en compte des aspects relatifs à la sécurité. Les problèmes posés concernent notamment l'intégrité, la confidentialité et le déni de service. Nous n'avons pour le moment pas traité cette question au cours de nos travaux. Si des techniques classiques de chiffrement doivent pouvoir être utilisées pour chiffrer, au niveau de l'intergiciel de communication, les données transmises, le problème de la distribution des clés de chiffrement reste à résoudre car cette distribution ne peut pas, comme cela est fait dans des réseaux d'infrastructure, reposer sur des entités de confiance stables (e.g. PKI : *Public Key Infrastructure*). Il est à noter que nos travaux sur les services pour MANET discontinus ont été réalisés notamment dans le cadre du projet ANR Sarah (décembre 2006 – juin 2009), projet qui comprenait un volet sur la sécurité couvert par nos collègues du XLIM et du LABRi. Les pistes qui ont été explorées pour résoudre les

problèmes de sécurité reposaient soit sur l'emploi de cartes à puces, soit sur la dérivation de clés de chiffrement asymétriques à partir des identités des utilisateurs. D'autres pistes devraient encore être étudiées, comme par exemple celles inspirées de travaux sur la réputation dans les réseaux pair à pair sur Internet.

Modèles de programmation, API et langages

Jusqu'à très récemment, les MANET ont été abordés essentiellement sous un angle « réseau ». Toute une communauté s'est évertuée à mettre en place des mécanismes pour fournir de la connectivité IP au sein d'un MANET. L'hypothèse a été trop souvent faite qu'il suffisait ensuite de déployer les applications usuelles qui sont mises en œuvre au-dessus d'IP. Or une très grande part des applications ne sont pas exploitables telles quelles dans des réseaux très dynamiques comme les MANET discontinus. C'est particulièrement vrai pour les applications fondées sur l'utilisation de TCP qui supporte mal les déconnexions fréquentes et prolongées. Il semble nécessaire de réécrire bon nombre d'applications en s'appuyant sur des modèles de programmation adaptés. En outre, des applications inédites vont sûrement émerger. On a présenté dans ce mémoire l'approche orientée services qui propose un modèle a priori favorable au développement d'applications pour les MANET discontinus. Mais même dans ce cas, il apparaît indispensable de modifier les intergiciels courants mettant en œuvre des mécanismes de découverte et d'invocation, afin de prendre en compte le caractère asynchrone des communications.

Il me semble qu'en complément des travaux sur les protocoles sur lesquels on peut maintenant avoir une vue d'ensemble, de nouveaux efforts de recherche devraient être désormais entrepris pour étudier des modèles de programmation aptes à faciliter le développement d'applications pour des réseaux très dynamiques tels que les MANET discontinus. Il s'agirait de formaliser des modèles à base, par exemple, de composants, de services, ou de données partagées (ou fondés sur une nouvelle abstraction) en suivant une approche non plus seulement tournée vers la définition de protocoles mais élargies vers la spécification d'API de programmation suffisament riches mais qui restent simple à maîtriser, voire de langages, en lien avec la spécification d'un large ensemble d'applications réelles.

Bibliographie

Publications personnelles

Dans ce paragraphe sont listées les publications dont je suis co-auteur. La clé de chaque référence débute par une lettre indiquant la catégorie de publication (*j* pour article de revue, *i* pour conférence internationale, *n* pour conférence nationale et et *r* pour rapport). Les références concernant les travaux décrits dans ce document (publiés dans la période 2002–2010) ont une clé mise en gras.

Articles de revues

[**j1**] Nicolas Le Sommer, Salma Ben Sassi, Frédéric Guidec et Yves Mahéo : A Middleware Support for Location-Based Service Discovery and Invocation in Disconnected MANETs. *Studia Informatica Universalis*, 8(3):71-97, septembre 2010.

[**j2**] Frédéric Guidec, Nicolas Le Sommer et Yves Mahéo : Opportunistic Software Deployment in Disconnected Mobile Ad Hoc Networks. *International Journal of Handheld Computing Research (IJHCR)*, 1(1):24-42, janvier 2010.

[**j3**] Didier Hoareau et Yves Mahéo : Middleware Support for Ubiquitous Software Components. *Personal and Ubiquitous Computing (PUC)*, 12(2):167-178, février 2008.

[j4] Frédéric Guidec et Yves Mahéo : POM : une machine virtuelle parallèle incorporant des mécanismes d'observation. *Calculateurs Parallèles*, 7(2):101-118, juin 1995.

[j5] Françoise André, Olivier Chéron, Marc Le Fur, Yves Mahéo et Jean-Louis Pazat : Programmation des machines à mémoire distribuée par distribution des données : langages et compilateurs. *Technique et Science Informatiques (TSI)*, 12(5):563-596, octobre 1993.

Articles de conférences et workshops internationaux

[**i1**] Yves Mahéo et Romeo Said : Service Invocation over Content-Based Communication in Disconnected Mobile Ad Hoc Networks. *In Proceedings of the 24th International Conference on Advanced Information Networking and Applications (AINA'10)*, Perth, Australie, avril 2010, pages 503-510. IEEE CS.

[**i2**] Nicolas Le Sommer, Romeo Said et Yves Mahéo : A Proxy-based Model for Service Provision in Opportunistic Networks. *In Proceedings of the 6th International Workshop on Middleware for Pervasive and Ad-Hoc Computing, 9th International Middleware Conference (Middleware 2008)*, Louvain, Belgique, décembre 2008. ACM.

[i3] Yves Mahéo, Romeo Said et Frédéric Guidec : Middleware Support for Delay-Tolerant Service Provision in Disconnected Mobile Ad Hoc Networks. *In Proceedings of the Workshop on Java and Components for Parallelism, Distribution and Concurrency, International Parallel and Distributed Processing Symposium (IPDPS'08)*, Miami, FL, USA, avril 2008. IEEE CS.

[i4] Romeo Said et Yves Mahéo : Toward a Platform for Service Discovery and Invocation in Disconnected Mobile Ad Hoc Networks. *In Proceedings of the 5th International Conference on Embedded and Ubiquitous Computing (EUC 2008)*, Shanghai, Chine, décembre 2008, pages 238-244. IEEE CS.

[i5] Frédéric Guidec et Yves Mahéo : Opportunistic Content-Based Dissemination in Disconnected Mobile Ad Hoc Networks. *In Proceedings of the International Conference on Mobile Ubiquitous Computing, Systems, Services and Technologies (UBICOMM 2007)*, Papeete, Tahiti, Polynésie française, novembre 2007, pages 49-54. IEEE CS.

[i6] Didier Hoareau, Takoua Abdellatif et Yves Mahéo : Architecture-Based Autonomic Deployment of J2EE Systems in Grids. *In Proceedings of the International Conference on Grid and Pervasive Computing (GPC'07)*, Paris, France, mai 2007, volume 4459 de *LNCS*, pages 362-373. Springer.

[i7] Takoua Abdellatif, Didier Hoareau et Yves Mahéo : Automated Deployment of Enterprise Systems in Large-Scale Environments. *In Proceedings of the 8th International Symposium on Distributed Objects and Applications (DOA'06)*, Montpellier, France, novembre 2006, volume 4277 de *LNCS*, pages 30-31. Springer.

[i8] Didier Hoareau et Yves Mahéo : Constraint-Based Deployment of Distributed Components in a Dynamic Network. *In Architecture of Computing Systems (ARCS 2006)*, Frankfurt/Main, Allemagne, mars 2006, volume 3864 de *LNCS*, pages 450-464. Springer.

[i9] Didier Hoareau et Yves Mahéo : Ubiquitous Fractal Components. *In Proceedings of the 5th Fractal Workshop, 20th European Conference on Object-Oriented Programming (ECOOP 2006)*, Nantes, France, juillet 2006.

[i10] Didier Hoareau et Yves Mahéo : Propagative Deployment of Hierarchical Components in a Dynamic Network. *In Proceedings of the 3rd International Working Conference on Component Deployment (CD 2005)*, Grenoble, France, novembre 2005, volume 3798 de *LNCS*, pages 111-114. Springer.

[i11] Didier Hoareau et Yves Mahéo : Distribution of a Hierarchical Component in a Non-Connected Environment. *In Proceedings of the 31st Euromicro Conference – Component-Based Software Engineering Track*, Porto, Portugal, septembre 2005, pages 143-150. IEEE CS.

[i12] Yves Mahéo, Frédéric Guidec et Luc Courtrai : Middleware Support for the Deployment of Resource-Aware Parallel Java Components on Heterogeneous Distributed Platforms. *In Proceedings of the 30th Euromicro Conference – Component-Based Software Engineering Track*, Rennes, France, septembre 2004, pages 144-151. IEEE CS.

[i13] Yves Mahéo, Frédéric Guidec et Luc Courtrai : Towards Resource-Aware Parallel Components. *In Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA'04)*, Las Vegas, NV, USA, juin 2004, pages 1006-1012. CSREA Press.

[**i14**] Yves Mahéo, Frédéric Guidec et Luc Courtrai : A Java Middleware Platform for Resource-Aware Distributed Applications. *In Proceedings of the 2nd International Symposium on Parallel and Distributed Computing (ISPDC'2003)*, Ljubljana, Slovénie, octobre 2003, pages 96-103. IEEE CS.

[**i15**] Luc Courtrai, Frédéric Guidec, Nicolas Le Sommer et Yves Mahéo : Resource Management for Parallel Adaptive Components. *In Proceedings of the Workshop on Java for Parallel and Distributed Computing, International Parallel and Distributed Processing Symposium (IPDPS'03)*, Nice, France, avril 2003, pages 134-141. IEEE CS.

[i16] Luc Courtrai, Yves Mahéo et Frédéric Raimbault : Java Objects Communication on a High Performance Network. *In Proceedings of the 9th Euromicro Workshop on Parallel and Distributed Processing (PDP 2001)*, Mantova, Italie, février 2001. IEEE CS.

[i17] Luc Courtrai, Yves Mahéo et Frédéric Raimbault : Expresso: A Library for Fast Transfers of Java Objects. *In Proceedings of the First Myrinet User Group Conference*, Lyon, France, septembre 2000, pages 43-49. INRIA.

[i18] Luisa Massari et Yves Mahéo : Performance Evaluation of Automatically Generated Data Parallel Programs. *In Proceedings of the 4th Euromicro Workshop on Parallel and Distributed Processing (PDP'96)*, Braga, Portugal, janvier 1996, pages 534-540. IEEE CS.

[i19] Françoise André et Yves Mahéo : CIDRE: A Distributed Shared Arrays Library. *In Proceedings of the 6th Workshop on Compilers for Parallel Computers (WPC'96)*, Aachen, Allemagne, décembre 1996, volume 21 de *Konferenzen des Forschungszentrums Jülich*, pages 1-8. Forschungszentrum Jülich.

[i20] Françoise André et Yves Mahéo : Cidre: Programming with Distributed Shared Arrays. *In Proceedings of the 3rd International Conference on High-Performance Computing (HiPC'96)*, Trivandrum, Inde, décembre 1996, pages 439-444. IEEE CS.

[i21] Françoise André, Marc Le Fur, Yves Mahéo et Jean-Louis Pazat : Parallelization of a Wave propagation Application using a Data Parallel Compiler. *In Proceedings of the 9th International Parallel Processing Symposium (IPPS'95)*, Santa Barbara, CA, USA, avril 1995, pages 760-765. IEEE CS.

[i22] Françoise André, Marc Le Fur, Yves Mahéo et Jean-Louis Pazat : The Pandore Data Parallel Compiler and its Portable Runtime. *In Proceedings of the International Conference and Exhibition on High-Performance Computing and Networking (HPCN Europe'95)*, Milan, Italie, mai 1995, volume 919 de *LNCS*, pages 176-183. Springer.

[i23] Frédéric Guidec et Yves Mahéo : POM: a Parallel Observable Machine. *In Parallel Computing: State of the Art and Perspectives, Proceedings of the Parallel Computing Conference (ParCo'95)*, Gand, Belgique, juin 1995, volume 11 de *Advances in Parallel Computing*, pages 343-350. Elsevier.

[i24] Frédéric Guidec et Yves Mahéo : POM: a Virtual Parallel Machine Featuring Observation Mechanisms. *In Proceedings of the 2nd International Conference on High-Performance Computing (HiPC'95)*, New Delhi, Inde, décembre 1995. Tata McGraw-Hill.

[i25] Marc Le Fur et Yves Mahéo : Efficient Communications in Parallel Loop Distribution. *In Parallel Computing: State of the Art and Perspectives, Proceedings of the Parallel Computing Conference (ParCo'95)*, Gand, Belgique, juin 1995, volume 11 de *Advances in Parallel Computing*, pages 359-366. Elsevier.

[i26] Yves Mahéo et Jean-Louis Pazat : Distributed Array Management for HPF Compilers. *In Proceedings of the 9th International Symposium on High Performance Computing Systems (HPCS'95)*, Montreal, Canada, juillet 1995, pages 119-129. Centre de Recherche Informatique de Montreal.

[i27] Yves Mahéo et Jean-Louis Pazat : Distributed Array Management Scheme for Data-parallel Compilers. *In Proceedings of the 5th Workshop on Compilers for Parallel Computers (WPC'95)*, Malaga, Espagne, mai 1995, pages 367-379.

[i28] Cyrille Bareau, Yves Mahéo et Jean-Louis Pazat : Parallel Program Performance Debugging with the Pandore II Environment. *In Parallel Computing: State of the Art and Perspectives, Proceedings of the Parallel Computing Conference (ParCo'93)*, Grenoble, France, mai 1993, volume 9 de *Advances in Parallel Computing*, pages 241-248. Elsevier.

Articles de conférences et workshops nationaux

[**n1**] Didier Hoareau et Yves Mahéo : Distribution d'un composant hiérarchique dans un environnement partiellement connecté. *In Actes des Journées Composants 2005*, Le Croisic, France, avril 2005, pages 103-111.

[**n2**] Luc Courtrai, Frédéric Guidec et Yves Mahéo : Gestion de ressources pour composants parallèles adaptables. *In Actes des Journées Composants, Systèmes à composants adaptables et extensibles*, Grenoble, France, octobre 2002, pages 97-109.

[**n3**] Luc Courtrai, Frédéric Guidec et Yves Mahéo : Concerto : gestion de ressources pour composants parallèles adaptables. *In Chapitre 4 des actes de l'École GRID'2002*, Aussois, France, décembre 2002, pages 41-53.

[n4] Luc Courtrai, Frédéric Guidec et Yves Mahéo : Concerto : composants parallèles adaptables. *In Actes des Journées Composants, Flexibilité du système au langage*, Besançon, France, octobre 2001, pages 155-156.

[n5] Françoise André et Yves Mahéo : CIDRE : une bibliothèque pour la distribution de tableaux partagés. *In Actes des journées sur la mémoire partagée répartie*, Bordeaux, France, mai 1996.

[n6] Françoise André et Yves Mahéo : CIDRE : une bibliothèque pour la distribution de tableaux partagés. *In Actes des 8e Rencontres Francophones sur le Parallélisme (RenPar'8)*, Bordeaux, France, mai 1996, pages 5-8.

[n7] Yves Mahéo : Évaluation de performances dans l'environnement Pandore II. *In Actes des 5e rencontres sur le parallélisme (RenPar'93)*, Brest, France, mai 1993, pages 47-50.

Rapports

[**r1**] Yves Mahéo, Luc Courtrai et Frédéric Guidec : Concerto : composants parallèles adaptables. Rapport de fin de projet ACI-GRID 2001 : JE2-Concerto, Valoria, Université de Bretagne-Sud, décembre 2003.

[r2] Yves Mahéo : *Environnements pour la compilation dirigée par les données : supports d'exécution et expérimentations*. Thèse de doctorat, IFSIC, Université de Rennes I, juillet 1995.

[r3] Yves Mahéo : Parallel Implementation of Gamma on a Shared Memory Architecture. Mémoire de DEA, IFSIC, Université de Rennes 1, Computer Science Laboratory, University of Newcastle Upon Tyne, septembre 1990.

Références

[1] Takoua Abdellatif, Jakub Kornas et Jean-Bernard Stefani : Reengineering J2EE Servers for Automated Management in Distributed Environments. *IEEE Distributed Systems Online*, 8(11), 2007.

[2] William Adjie-Winoto, Elliot Schwartz, Hari Balakrishnan et Jeremy Lilley : The Design and Implementation of an Intentional Naming System. *In Proceedings of the 17th ACM Symposium on Operating System Principles (SOSP'99)*, Kiawah Island Resort, near Charleston, SC, USA, décembre 1999, pages 186-201. ACM.

[3] Marco Aldinucci, Sonia Campa, Massimo Coppola, Marco Danelutto, Domenico Laforenza, Diego Puppin, Luca Scarponi, Marco Vanneschi et Corrado Zoccolo : Components for high performance grid programming in the Grid.it project. *In Proceedings of the Workshop on Component Models and Systems for Grid Applications*, Saint Malo, France, juin 2004, pages 19-38. Springer.

[4] Benjamin A. Allan, Robert Armstrong, David E. Bernholdt, Felipe Bertrand, Kenneth Chiu, Tamara L. Dahlgren, Kostadin Damevski, Wael R. Elwasif, Thomas G. W. Epperly, Madhusudhan Govindaraju, Daniel S. Katz, James A. Kohl, Manoj Krishnan, Gary Kumfert, J. Walter Larson, Sophia Lefantzi, Michael J. Lewis, Allen D. Malony, Lois C. Mclnnes, Jarek Nieplocha, Boyana Norris, Steven G. Parker, Jaideep Ray, Sameer Shende, Theresa L. Windus et Shujia Zhou : A Component Architecture for High-Performance Scientific Computing. *International Journal of High Performance Computing Applications*, 20(2):163-202, 2006.

[5] Robert Allen : *A Formal Approach to Software Architecture*. Thèse de doctorat, School of Computer Science, Carnegie Mellon University, mai 1997. CMU-CS-97-144.

[6] Ken Arnold, éditeur. *The Jini(TM) Specifications*. Addison-Wesley Professional, 2nde édition, 2000.

[7] Godmar Back, Wilson C. Hsieh et Jay Lepreau : Processes in KaffeOS: Isolation, Resource Management, and Sharing in Java. *In Proceedings of the 4th Symposium on Operating Systems Design and Implementation*, San Diego, CA, USA, octobre 2000.

[8] Godmar Back, Patrick Tullmann, Leigh Stoller, Wilson C. Hsieh et Jay Lepreau : Techniques for the Design of Java Operating Systems. *In Proceedings of the USENIX Annual Technical Conference*, San Diego, CA, USA, juin 2000.

[9] Laurent Baduel, Françoise Baude, Denis Caromel, Arnaud Contes, Fabrice Huet, Matthieu Morel et Romain Quilici : *Grid Computing: Software Environments and Tools*, chapitre Programming, Deploying, Composing, for the Grid. Springer, janvier 2006.

[10] Roberto Baldoni, Marin Bertier, Michel Raynal et Sara Tucci Piergiovanni : Looking for a Definition of Dynamic Distributed Systems. *In Proceedings of the 9th International Conference on Parallel Computing Technologies (PaCT'07)*, Pereslavl-Zalessky, Russie, septembre 2007, volume 4671 de *LNCS*, pages 1-14. Springer.

[11] Lionel Barrère, Arnaud Casteigst et Serge Chaumette : A Totally Decentralized Document Sharing System for Mobile Ad Hoc Networks. *In Proceedings of the 4th ACM International Workshop on Mobility Management and Wireless Access (MobiWac 2006)*, Malaga, Espagne, octobre 2006, pages 116-120. ACM.

[12] Françoise Baude, Denis Caromel et Matthieu Morel : From Distributed Objects to Hierarchical Grid Components. *In Proceedings of the International Symposium on Distributed Objects and Applications (DOA'2003)*, Catania, Sicile, novembre 2003, volume 2888 de *LNCS*, pages 1226-1242. Springer.

[13] Abdelkrim Beloued, Chantal Taconet, Dhouha Ayed et Guy Bernard : Placement automatique des composants lors du déploiement d'applications à base de composants. *In Journée Composants (JC 05)*, Le Croisic, France, avril 2005.

[14] Salma Ben Sassi et Nicolas Le Sommer : Towards an Opportunistic and Location-Aware Service Provision in Disconnected Mobile Ad Hoc Networks. *In Proceedings of the 2nd International ICST Conference on Mobile Wireless Middleware, Operating Systems, and Applications (Mobilware 2009)*, Berlin, Allemagne, avril 2009, volume 7 de *LNICST*, pages 396-406. Springer.

[15] Salma Ben Sassi et Nicolas le Sommer : Une plate-forme intergicielle pour la découverte et l'invocation de services géolocalisés dans les réseaux ad hoc discontinus. *In Actes de la 9e Conférence Internationale sur les Nouvelles Technologies de la Répartition (Notere 2009)*, Montréal, Canada, juillet 2009, pages 28-37. Université de Québec à Montréal.

[16] Francine Berman, Andrew Chien, Keith Cooper, Jack Dongarra, Ian Foster, Dennis Gannon, Lennart Johnsson, Ken Kennedy, Carl Kesselman, John Mellor-Crumme, Dan Reed, Linda Torczon et Rich Wolski : The GrADS Project: Software Support for High-Level Grid Application Development. *International Journal of High Performance Computing Applications*, 15:327-344, 2003.

[17] Bluetooth Special Interest Group : Specification of the Bluetooth System, Version 2.1 + EDR, juillet 2007. `http://www.bluetooth.com`.

[18] Frederic Briclet, Christophe Contreras et Philippe Merle : OpenCCM : une infrastructure a composants pour le déploiement d'applications à base de composants CORBA. *In Actes de la 1re conférence Francophone sur le Déploiement et la (Re)-Configuration de Logiciels (Decor 2004)*, Grenoble, France, juin 2004. Net Print, Eybens.

[19] Éric Bruneton, Thierry Coupaye, Matthieu Leclercq, Vivien Quéma et Jean-Bernard Stefani : An Open Component Model and its Support in Java. *In Proceedings of the International Symposium on Component-based Software Engineering (CBSE7)*, Édimbourg, Écosse, mai 2004, volume 3054 de *LNCS*. Springer.

[20] Éric Bruneton, Thierry Coupaye et Jean-Bernard Stefani : The Fractal Component Model. OW2 Consortium Specification, version 2.0-3, février 2004. `http://fractal.ow2.org`.

[21] Jérémy Buisson, Françoise André et Jean-Louis Pazat : Dynamic Adaptation for Grid Computing. *In Advances in Grid Computing – European Grid Conference (Revised Selected Papers)*, Amsterdam, Pays-Bas, juin 2005, volume 3470 de *LNCS*, pages 538-547. Springer.

[22] Éric Cariou : *Contribution à un processus de réification d'abstractions de communication*. Thèse de doctorat, Université de Rennes 1, juin 2003.

[23] Antonio Carzaniga, Alfonso Fuggetta, Richard S. Hall, Dennis Heimbigner, André van der Hoek et Alexander L. Wolf : A Characterization Framework for Software Deployment Technologies. Rapport technique CU-CS-857-98, Department of Computer Science, University of Colorado, avril 1998.

[24] Arnaud Casteigts, Serge Chaumette et Afonso Ferreira : Characterizing topological assumptions of distributed algorithms in dynamic networks. *In Proceedings of the 16th International Conference on Structural Information and Communication Complexity (SIROCCO'09)*, Piran, Slovénie, mai 2009, volume 5869 de *LNCS*, pages 126-140. Springer.

[25] Emmanuel Cecchet, Hazem Elmeleegy, Oussama Layaida et Vivien Quema : Implementing Probes for J2EE Cluster Monitoring. *Studia Informatica*, 4(1):31-40, 2005.

[26] Vinton Cerf, Scott Burleigh, Adrian Hooke, Leigh Torgerson, Robert Durst, Keith Scott, Kevin Fall et Howard Weiss : Delay-Tolerant Networking Architecture. IETF RFC 4838, avril 2007.

[27] Ian Chakeres et Charles Perkins : Dynamic MANET On-demand (DYMO) Routing. IETF, Internet Draft draft-ietf-manet-dymo-17, mars 2009.

[28] Dipanjan Chakraborty, Anupam Joshi et Yelena Yesha : Integrating service discovery with routing and session management for ad-hoc networks. *Ad Hoc Networks*, 4(2):204-224, mars 2006.

[29] Dipanjan Chakraborty, Anupam Joshi, Yelena Yesha et Tim Finin : Toward distributed service discovery in pervasive computing environments. *IEEE Transactions on Mobile Computing*, 5(2):97-112, février 2006.

[30] Fangzhe Chang, Ayal Itzkovitz et Vijay Karamcheti : User-level Resource-constrained Sandboxing. *In Proceedings of the 4th USENIX Windows Systems Symposium*, Seattle, WA, USA, août 2000. USENIX Association.

[31] Steve J. Chapin, Dimitrios Katramatos, John Karpovich et Andrew S. Grimshaw : The Legion Resource Management System. *In Proceedings of the 5th Workshop on Job Scheduling Strategies for Parallel Processing, International Parallel Processing Symposium (IPPS'99)*, San Juan, Puerto Rico, avril 1999, volume 1659 de *LNCS*. Springer.

[32] Chunglae Cho et Duccki Lee : Survey of Service Discovery Architectures for Mobile Ad hoc Networks. Term paper, Mobile Computing, CEN 5531, Dept. Computer and Information Science and Eng., Univ. Florida, Fall, 2005.

[33] Thomas Clausen, Christopher Dearlove et Philippe Jacquet : The Optimized Link State Routing Protocol version 2. IETF, Internet Draft draft-ietf-manet-olsrv2-10, septembre 2009.

[34] Denis Conan, Romain Rouvoy et Lionel Seinturier : COSMOS : composition de nœuds de contexte. *Technique et Science Informatiques*, 27(10):1189-1224, 2008.

[35] Marco Conti et Mohan Kumar : Opportunities in Opportunistic Computing. *IEEE Computer*, 43:42-50, 2010.

[36] Paolo Costa, Mirco Musolesi, Cecilia Mascolo et Gian Pietro Picco : Adaptive Content-based Routing for Delay-tolerant Mobile Ad Hoc Networks. Rapport technique, University College London, août 2006.

[37] Grzegorz Czajkowski et Thorsten von Eicken : JRes: a Resource Accounting Interface for Java. *In Proceedings of the ACM SIGPLAN Conference on Object-Oriented Programming Systems, Languages & Applications (OOPSLA'98)*, Vancouver, Canada, 1998.

[38] Karl Czajkowski, Steven Fitzgerald, Ian Foster et Carl Kesselman : Grid Information Services for Distributed Resource Sharing. *In Proceedings of the 10th IEEE International on High Performance Distributed Computing (HPDC'01)*, San Francisco, CA, USA, août 2001. IEEE CS.

[39] Anwitaman Datta, Silvia Quarteroni et Karl Aberer : Autonomous Gossiping: a Self-Organizing Epidemic Algorithm for Selective Information Dissemination in Mobile Ad-Hoc Networks. *In Proceedings of the International Conference on Semantics of a Networked World (IC-SNW'04)*, Paris, France, juin 2004, volume 3226 de *LNCS*, pages 126-143. Springer.

[40] Alan Dearle, Graham N. C. Kirby et Andrew J. McCarthy : A Framework for Constraint-Based Deployment and Autonomic Management of Distributed Applications. *In Proceedings of the First International Conference on Autonomic Computing (ICAC 2004)*, New-York, NY, USA, mai 2004, pages 300-301. IEEE CS.

[41] Alexandre Denis, Christian Pérez, Thierry Priol et André Ribes : Padico: A Component-Based Software Infrastructure for Grid Computing. *In Proceedings of the 17th International Parallel and Distributed Processing Symposium (IPDPS'2003)*, Nice, France, avril 2003. IEEE CS.

[42] DMTF : CIM System Virtualization Model. Rapport technique White Paper version 1.0.0, Distributed Management Task Force, novembre 2007. `http://www.dmtf.org`.

[43] Jack Dongarra et Victor Eijkhout : Self-Adapting Numerical Software for Next Generation Applications. *International Journal of High Performance Computing Applications*, 17:125-131, 2003.

[44] Avri Doria, Maria Uden et Durga Prasad Pandey : Providing connectivity to the Saami nomadic community. *In 2nd International Conference on Open Collaborative Design for Sustainable Innovation*, Bangalore, Inde, décembre 2002.

[45] Jérémy Dubus : *Une démarche orienté modèle pour le déploiement de systèmes en environnements ouverts distribués*. Thèse de doctorat, Université des Sciences et Technologies de Lille (USTL), octobre 2008.

[46] Wolfgang Emmerich : Software Engineering and Middleware: a Roadmap. *In Proceedings of the 22nd International Conference on Software Engineering (ICSE'00) – Future of SE Track*, Limerick, Irlande, 2000, pages 117-129. ACM.

[47] Paal E. Engelstad, Yan Zheng, Rajeev Koodli et Charles E. Perkins : Service Discovery Architectures for On-Demand Ad Hoc Networks. *International Journal of Ad Hoc and Sensor Wireless Networks*, 2(1):27-58, 2006.

[48] Patrick Eugster, Pascal Felber, Rachid Guerraoui et Anne-Marie Kermarrec : The Many Faces of Publish/Subscribe. *ACM Computing Surveys*, 35(2):114-131, 2003.

[49] David Evans et Andrew Twyman : Flexible Policy-Directed Code Safety. *In Proceedings of the 20th IEEE Symposium on Security and Privacy*, Oakland, CA, USA, mai 1999. IEEE CS.

[50] Kevin Fall, Wei Hong et Samuel Madden : Custody Transfer for Reliable Delivery in Delay Tolerant Networks. Rapport technique, Intel Research Berkeley, 2003.

[51] Afonso Ferreira : Building a reference combinatorial model for MANETs. *IEEE Network*, 18(5):24-29, 2004.

[52] Michael J. Fisher, Nancy A. Lynch et Michael S. Patterson : Impossibility of distributed consensus with one faulty process. *Journal of the ACM*, 32(2):374-382, 1985.

[53] Jean-Patrick Gelas : *Vers la conception d'une architecture de réseaux actifs*. Thèse de doctorat, Université Claude Bernard Lyon 1, décembre 2003.

[54] Globus Alliance : The Globus Toolkit. `http://www.globus.org/toolkit`.

[55] Erik Guttmann, Charles Perkins, Veizades John et Michael Day : Service Location Protocol, Version 2. IETF RFC 2608, mars 1999.

[56] Julien Haillot et Frédéric Guidec : A Protocol for Content-Based Communication in Disconnected Mobile Ad Hoc Networks. *In Proceedings of the 22nd International Conference on Advanced Information Networking and Applications (AINA'08)*, Okinawa, Japon, mars 2008, pages 188-195. IEEE CS.

[57] Julien Haillot et Frédéric Guidec : Content-Based Communication in Disconnected Mobile Ad Hoc Networks. *In Actes de la 8e Conférence Internationale sur les Nouvelles Technologies de la Répartition (Notere 2008)*, Lyon, France, juin 2008, pages 162-173. ACM.

[58] Julien Haillot et Frédéric Guidec : Towards a Usenet-like Discussion System for Users of Disconnected MANETs. *In Proceedings of the 1st International Workshop on Opportunistic Networking (WON'08)*, Okinawa, Japon, mars 2008, pages 1678-1683. IEEE CS.

[59] Julien Haillot et Frédéric Guidec : A Protocol for Content-Based Communication in Disconnected Mobile Ad Hoc Networks. *Mobile Information Systems*, 6(2):123-154, juin 2010.

[60] Julien Haillot, Frédéric Guidec, Serge Corlay et Jacques Turbert : Disruption-Tolerant Content-Driven Information Dissemination in Partially Connected Military Tactical Radio Networks. *In Proceedings of the 28th Military Communication Conference (MILCOM'2009)*, Boston, USA, octobre 2009. IEEE CS.

[61] Gertjan P. Halkes, Aline Baggio et Koen G. Langendoen : A Simulation Study of Integrated Service Discovery. *In 1st European Conference on Smart Sensing and Context (EuroSCC 2006)*, Enschede, Pays-Bas, octobre 2006, volume 4272 de *LNCS*, pages 39-53. Springer.

[62] Richard S. Hall : *Agent-based Software Configuration and Deployment*. Thèse de doctorat, University of Colorado at Boulder, décembre 1999.

[63] Radu Handorean, Rohan Sen, Greg Hackmann et Gruia-Catalin Roman : Context Aware Session Management for Services in Ad Hoc Networks. *In Proceedings of the International Conference on Services Computing (SCC'05)*, Orlando, FL, USA, juillet 2005, volume 1, pages 113-120. IEEE CS.

[64] Khaled A. Harras, Kevin C. Almeroth et Elisabeth M. Belding-Royer : Delay Tolerant Mobile Networks (DTMNs): Controlled Flooding in Sparse Mobile Networks. *In Proceedings of the IFIP Networking Conference*, Waterloo, Canada, mai 2005.

[65] Sumi Helal, Nitin Desai, Varun Verma et Choonhwa Lee : Konark: Service Discovery and Delivery Protocol for Ad-hoc Networks. *In Proceedings of the 3rd IEEE Conference on Wireless Communication Networks (WCNC)*, La Nouvelle-Orléans, LA, USA, mars 2003.

[66] Reto Hermann, Dirk Husemann, Michael Moser, Michael Nidd, Christian Rohner et Andreas Schade : DEAPspace – Transient ad hoc networking of pervasive devices. *Computer Networks*, 35(4):411-428, mars 2001.

[67] Charles Antony Richard Hoare : *Communicating Sequential Processes*. International Series in Computer Science. Prentice Hall, 1985.

[68] Didier Hoareau : *Composants ubiquitaires pour réseaux dynamiques*. Thèse de doctorat, Université de Bretagne-Sud, décembre 2007.

[69] Didier Hoareau et Chouki Tibermacine : Component Deployment Evolution Driven by Architectural Patterns and Resource Requirements. *In Proceedings of the 3rd European Workshop on Software Architectures, Languages, Styles, Models, Tools, and Applications (EWSA'06)*, Nantes, France, septembre 2006, volume 4344 de *LNCS*, pages 236-243. Springer.

[70] Todd D. Hodes, Steven E. Czerwinski, Ben Y. Zhao, Anthony D. Joseph et Randy H. Katz : An architecture for secure wide-area service discovery. *Wireless Networks*, 8(2/3):213-230, 2002.

[71] Andreas Hoffmann, Marc Born, Christophe Contreras, Bertram Neubauer et Bertil Folliot : Deployment and Configuration of Component Assemblies. Rapport technique 4.2, IST COACH document & logiciel livrable, octobre 2003. `http://www.ist-coach.org`.

[72] Luc Hogie, Pascal Bouvry et Frédéric Guinand : The MADHOC simulator. `http://www-lih.univ-lehavre.fr/~hogie/madhoc`.

[73] Seyed Amin Hosseini Seno, Rahmat Budiarto et Tat-Chee Wan : Survey and New Approach in Service Discovery and Advertisement for Mobile Ad hoc Networks. *International Journal of Computer Science and Network Security*, 7(2):275-284, 2007.

[74] Pan Hui, Jon Crowcroft et Eiko Yoneki : BUBBLE Rap: Social Based Forwarding in Delay Tolerant Networks. *In Proceedings of the 9th ACM International Symposium on Mobile Ad Hoc Networking and Computing (MobiHoc)*, Hong Kong, Chine, mai 2008, pages 241-250. ACM.

[75] Information Technology : Telecommunications and Information Exchange between Systems, Local and Metropolitan Area Networks, Specific Requirements Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications. ANSI/IEEE Std 802.11, 1999.

[76] Information Technology : Telecommunications and Information Exchange between Systems, Local and Metropolitan Area Networks, Specific Requirements Part 15: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications for Wireless Personal Area Networks (WPANs). ANSI/IEEE Std 802.15, 2002.

[77] Information Technology : Telecommunications and Information Exchange between Systems, Local and Metropolitan Area Networks, Specific Requirements Part 16: Air Interface for Fixed Broadband Wireless Access Systems. ANSI/IEEE Std 802.16, 2004.

[78] Java Community Process : Java EE Application Deployment, version 2.1. `http://jcp.org/en/jsr/detail?id=88`, juillet 2002.

[79] Jose Luis Jodra, Maribel Vara, Jose Ma Cabero et Josu Bagazgoitia : Service Discovery Mechanism Over OLSR for Mobile Ad-hoc Networks. *In Proceedings of the International Conference on Advanced Information Networking and Applications (AINA 2006)*, Vienne, Autriche, avril 2006, volume 2, pages 534-542. IEEE CS.

[80] Philo Juang, Hidekazu Oki, Yong Wang, Margaret Martonosi, Li Shiuan Peh et Daniel Rubenstein : Energy-efficient computing for wildlife tracking: design tradeoffs and early experiences with ZebraNet. *ACM SIGARCH Computer Architecture News (Proceedings of the 10th annual conference on Architectural Support for Programming Languages and Operating Systems)*, 30(5):96-107, 2002.

[81] Patrick Kellert et Farouk Toumani : *Les Web services sémantiques*. Revue I3 (Information, Interaction, Intelligence). Hors série « Le web sémantique ». Cépaduès Éditions, 2004.

[82] Michael Klein, Birgitta König-Ries et Obreiter Philipp : Lanes: A Lightweight Overlay for Service Discovery in Mobile Ad Hoc Networks. *In Proceedings of the 3rd Workshop on Applications and Services in Wireless Networks (ASWN 03)*, Berne, Suisse, juillet 2003.

[83] Michael Klein, Birgitta König-Ries et Obreiter Philipp : Service Rings – A Semantic Overlay for Service Discovery in Ad hoc Networks. *In Proceedings of the International Workshop on Database and Expert Systems Applications (DEXA'03)*, Los Alamitos, CA, USA, 2003, pages 180-185. IEEE CS.

[84] Sacha Krakowiak, Thierry Coupaye, Vivien Quéma, Lionel Seinturier, Jean-Bernard Stefani, Marc Dumas, Marie-Christine Fauvet, Pascal Déchamboux, Michel Riveill, Antoine Beugnard, David Emsellem, Didier Donsez, Ahmed Aït-Bachir, Roland Balter, André Freyssinet, Yoann Bersihand et Sébastien Chassande-Barrioz : Intergiciel et Construction d'Applications Réparties. `http://sardes.inrialpes.fr/ecole/livre/pub`, 2007.

[85] Klaus Krauter, Rajkumar Buyya et Muthucumaru Maheswaran : A Taxonomy and Survey of Grid Resource Management Systems for Distributed Computing. *Sofware – Practice and Experience*, 32(2):135-164, février 2002.

[86] Stuart Kurkowski, Tracy Camp et Michael Colagrosso : MANET simulation studies: the incredibles. *ACM SIGMOBILE Mobile Computing and Communications Review*, 9(4):50-61, 2005.

[87] Sébastien Lacour : *Contribution à l'automatisation du déploiement d'applications sur des grilles de calcul*. Thèse de doctorat, Université de Rennes I, décembre 2005.

[88] Kung-Kiu Lau et Zheng Wang : Software component models. *IEEE Transactions on Software Engineering*, 33:709-724, 2007.

[89] Nicolas Le Sommer : A Framework for Service Provision in Intermittently Connected Mobile Ad hoc Networks. *In Proceedings of the 8th International Symposium on a World of Wireless, Mobile and Multimedia Networks (WOWMOM 2007)*.

[90] Nicolas Le Sommer : Service Provision in Disconnected Mobile Ad Hoc Networks. *In Proceedings of the International Conference on Mobile Ubiquitous Computing, Systems, Services and Technologies (UBICOMM 2007)*, Papeete, Polynésie Française, novembre 2007, pages 125-130. IEEE CS.

[91] Nicolas Le Sommer : Vers une ubiquité d'accès aux services dans les réseaux mobiles ad hoc. *In 7e Conférence Internationale sur les NOuvelles TEchnologies de la REpartition (Notere 2007)*, pages 207-218. Hermes Sciences.

[92] Nicolas Le Sommer et Salma Ben Sassi : Location-based Service Discovery and Delivery in Opportunistic Networks. *In Proceedings of the 9th International Conference on Networks (ICN 2010)*, Les Ménuires, France, avril 2010, pages 179-184. IEEE CS.

[93] Laurent Lefèvre et Jean-Patrick Gelas : *Programmable Networks for IP Service Deployment*, chapitre 14, « High Performance Execution Environments », pages 291-321. Artech House Books, mai 2004. ISBN 1-58053-745-6.

[94] Jérémie Leguay, Timur Friedman et Vania Conan : DTN Routing in a Mobility Pattern Space. *In Proceedings of the ACM SIGCOMM Workshop on Delay Tolerant Networking and related topics (WDTN-05)*, Philadelphie, PA, USA, août 2005. ACM.

[95] Vincent Lenders, Martin May et Bernhard Plattner : Service Discovery in Mobile Ad Hoc Networks: A Field Theoretic Approach. *In Proceedings of the 6th IEEE International Symposium on a World of Wireless, Mobile, and Multimedia Networks (WoWMoM 2005)*, Taormina, Italie, juin 2005, pages 120-130. IEEE CS.

[96] Vincent Lestideau : *Modèles et environnement pour configurer et déployer des systèmes logiciels*. Thèse de doctorat, Université de Savoie, décembre 2003.

[97] Anders Lindgren, Avri Doria et Olov Schelén : Probabilistic Routing in Intermittently Connected Networks. *In Proceedings of the 1st International Workshop on Service Assurance with Partial and Intermittent Resources (SAPIR 2004)*, Fortaleza, Brésil, août 2004, volume 3126 de *LNCS*. Springer.

[98] Changling Liu et Jörg Kaiser : A Survey of Mobile Ad Hoc network Routing Protocols. Rapport technique, University of Magdeburg, 2005.

[99] Jeff Magee, Naranker Dulay, Susan Eisenbach et Jeff Kramer : Specifying Distributed Software Architectures. *In Proceedings of the 5th European Software Engineering Conference (ESEC'95)*, Sitges, Espagne, 1995, volume 989, pages 137-153. Springer.

[100] Jeff Magee, Naranker Dulay et Jeff Kramer : Regis: A Constructive Development Environment for Distributed Programs. *Distributed Systems Engineering Journal*, 1(5):304-312, 1994.

[101] Jeff Magee et Jeff Kramer : Specifying Distributed Software Architectures. *In Proceedings of the 4th ACM SIGSOFT Symposium on the Foundations of Software Engineering (FSE 4)*, San Francisco, CA, USA, 1996, numéro 6 de 21, pages 3-14. ACM.

[102] Marco Mamei et Franco Zambonelli : Programming Pervasive and Mobile Computing Applications: the TOTA Approach. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, 18(4):1-56, juillet 2009.

[103] Corinne Marchand : *Mise au points d'algorithmes répartis dans un environnement fortement variable, et expérimentation dans le contexte des pico-réseaux*. Thèse de doctorat, Institut National Polytechnique de Grenoble, décembre 2004.

[104] Cecilia Mascolo, Licia Capra, Stefanos Zachariadis et Wolfgang Emmerich : XMIDDLE: A Data-Sharing Middleware for Mobile Computing. *Personal and Wireless Communications Journal*, 21(1):77-103, avril 2002.

[105] Elton N. Mathias, Françoise Baude, Vincent Cavé et Nicolas Maillard : A Component-Oriented Support for Hierarchical MPI Programming on Multi-Cluster Grid Environments. *In Proceedings of the 19th International Symposium on Computer Architecture and High Performance Computing (SBAC-PAD 2007)*, Gramado, Brésil, octobre 2007, pages 135-142. IEEE CS.

[106] Doug McIlroy : Mass Produced Software Components. *In Software Engineering: Report of a conference sponsored by the NATO Science Committee*, Garmisch, Allemagne, 7-11 Oct. 1968, janvier 1969, pages 138-155. Scientific Affairs Division, NATO.

[107] Nenad Medvidovic et Richard N. Taylor : A Classification and Comparison Framework for Software Architecture Description Languages. *IEEE Transactions on Software Engineering*, 26(1):70-93, 2000.

[108] René Meier, Vinny Cahill, Andronikos Nedos et Siobhan Clarke : Proximity-Based Service Discovery in Mobile Ad Hoc Networks. *In Proceedings of the 5th IFIP International*

Conference on Distributed Applications and Interoperable Systems (DAIS'05), Athènes, Grèce, juin 2005, volume 3543 de *LNCS*. Springer.

[109] Bertrand Meyer : The Grand Challenge of Trusted Components. *In Proceedings of the 25th International Conference on Software Engineering (ICSE'03)*, Portland, OR, USA, mai 2003, pages 660-667. IEEE CS.

[110] Adnan Noor Mian, Roberto Baldoni et Roberto Beraldi : A Survey of Service Discovery Protocols in Multihop Mobile Ad Hoc Networks. *IEEE Pervasive Computing*, 8:66-74, 2009.

[111] Microsoft : .Net. `http://www.microsoft.com/net`.

[112] Marija Mikic-Rakic et Nenad Medvidovic : Software Architectural Support for Disconnected Operation in Highly Distributed Environments. *In Proceedings of the International Symposium on Component-based Software Engineering (CBSE7)*, Édimbourg, Écosse, mai 2004, volume 3054 de *LNCS*, pages 23-39. Springer.

[113] Richard Monson-Haefel et Bill Burke : *Enterprise JavaBeans 3.0 – Developing Enterprise Java Components*. O'Reilly Media, 5e édition, 2006.

[114] Achour Mostéfaoui, Sergio Rajsbaum, Michel Raynal et Matthieu Roy : Condition-based consensus solvability: a hierarchy of conditions and efficient protocols. *Distributed Computing*, 17(1):1-20, 2004.

[115] Amy L. Murphy, Gian Pietro Picco et Gruia-Catalin Roman : Lime: A Coordination Middleware Supporting Mobility of Hosts and Agents. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, 15(3):279-328, juillet 2006.

[116] Mirco Musolesi et Cecilia Mascolo : CAR: Context-Aware Adaptive Routing for Delay Tolerant Mobile Networks. *IEEE Transactions on Mobile Computing*, 8(2):246-260, 2009.

[117] Michael O. Neary, Alan Phipps, Steven Richman et Peter Capello : Javelin 2.0: Java-based Parallel Computing on the Internet. *In Proceedings of the European Parallel Computing Conference (Euro-Par'2000)*, Munich, Allemagne, août 2000, volume 1900 de *LNCS*, pages 1231-1238. Springer.

[118] Andronikos Nedos, Kulpreet Singh et Siobhan Clarke : Service*: Distributed Service Advertisement for Multi-Service, Multi-Hop MANET Environments. *In Proceedings of 7th IFIP International Conference on Mobile and Wireless Communication Networks (MWCN'05)*, Marrakech, Maroc, septembre 2005.

[119] Hoang Anh Nguyen et Silvia Giordano : Routing in Opportunistic Networks. *International Journal of Ambient Computing and Intelligence (IJACI)*, 1, 2009.

[120] Hoang Anh Nguyen, Silvia Giordano et Alessandro Puiatti : Probabilistic Routing Protocol for Intermittently Connected Mobile Ad hoc Network (PROPICMAN). *In International Symposium on a World of Wireless, Mobile and Multimedia Networks (WoWMoM 2007)*, Helsinky, Finland, juin 2007, pages 1-6. IEEE CS.

[121] OASIS : UDDI Version 3.0.2. UDDI Spec Technical Committee Draft, novembre 2004. `http://www.w3c.org`.

[122] Object Management Group : Trading Object Service Specification, Version 1.0, mai 2000. formal/2000-06-27.

[123] Object Management Group : Corba Component Model Specification, Version 4.0, avril 2006. formal/06-04-01.

[124] Object Management Group : Deployment and Configuration of Component-based Distributed Applications Specification, Version 4.0, avril 2006. formal/06-04-02.

[125] Stephan Olariu et Michele C. Weigle, éditeurs. *Vehicular Networks: From Theory to Practice*. Information Science Series. Chapman & Hall/CRC Computer, 1re édition, 2009.

[126] OSGI Alliance : OSGi Service Platform, Core Specification, Release 4, Version 4.2, septembre 2009. `http://www.osgi.org`.

[127] Guilhem Paroux, Ludovic Martin, Julien Nowalczyk et Isabelle Demeure : Transhumance: A power-sensitive middleware for data sharing on mobile ad hoc networks. *In 7th International Workshop on Applications and Services in Wireless Networks (ASWN'07)*, Santander, Espagne, mai 2007. Papier n°1569024426.

[128] Andrea Passarella, Mohan Kumar, Marco Conti et Eleonora Borgia : Minimum-Delay Service Provisioning in Opportunistic Networks. *IEEE Transactions on Parallel and Distributed Systems*, août 2010.

[129] Luciana Pelusi, Andrea Passarella et Marco Conti : Opportunistic Networking: Data Forwarding in Disconnected Mobile Ad Hoc Networks. *IEEE Communications Magazine*, 44(11):134-141, novembre 2006.

[130] Alex (Sandy) Pentland, Richard Fletcher et Amir Hasson : Daknet: Rethinking connectivity in developing nations. *Computer*, 37(1):78-83, janvier 2004.

[131] Vivien Quéma, Roland Balter, Luc Bellissard, David Féliot, André Freyssinet et Serge Lacourte : Asynchronous, Hierarchical and Scalable Deployment of Component-Based Applications. *In Proceedings of the 2nd International Working Conference on Component Deployment (CD'2004)*, Édimbourg, Écosse, mai 2004.

[132] Rajesh Raman, Miron Livny et Marvin Solomon : Matchmaking: Distributed Resource Management for High Throughput Computing. *In Proceedings of the 7th IEEE International Symposium on High Performance Distributed Computing*, Chicago, IL, USA, juillet 1998. IEEE CS.

[133] Jan S. Rellermeyer, Gustavo Alonso et Timothy Roscoe : R-OSGi: Distributed Applications through Software Modularization. *In Proceedings of the 8th International Middleware Conference Conference (Middleware 2007)*, Newport Beach, CA, USA, novembre 2007, volume 4834 de *LNCS*. Springer.

[134] Elisabeth M. Royer et Chai-Keong Toh : A Review of Current Routing Protocols for Ad-Hoc Mobile Wireless Networks. *IEEE Personal Communications*, pages 46-55, avril 1999.

[135] Françoise Sailhan et Valérie Issarny : Scalable Service Discovery for MANET. *In Proceedings of the 3rd International Conference on Pervasive Computing and Communications (PerCom'2005)*, Hawai, HI, USA, mars 2005. IEEE CS.

[136] Keith Scott et Scott Burleigh : Delay-Tolerant Networking Architecture. IETF RFC 5050, avril 2007.

[137] Tara Small et Zygmunt J. Haas : The shared wireless infostation model: a new ad hoc networking paradigm (or where there is a whale, there is a way). *In Proceedings of the 4th ACM International Symposium on Mobile Ad Hoc Networking & Computing (MobiHoc'03)*, Annapolis, MD, USA, 2003, pages 233-244. ACM.

[138] Thrasyvoulos Spyropoulos, Konstantinos Psounis et Cauligi S. Raghavendra : Spray and Wait: an Efficient Routing Scheme for Intermittently Connected Mobile Networks. *In Proceedings of the ACM SIGCOMM Workshop on Delay-Tolerant Networking (WDTN'05)*, Philadelphie, PA, USA, 2005, pages 252-259. ACM.

[139] Clemens Szyperski : *Component Software – Beyond Object-Oriented Programming*. Addison-Wesley and ACM Press, 2nde édition, 2002. ISBN 0-201-74572-0.

[140] Chouki Tibermacine, Didier Hoareau et Réda Kadri : Enforcing Architecture and Deployment Constraints of Distributed Component-based Software. *In Proceedings of the International Conference on Fundamental Approaches to Software Engineering (FASE'07)*, Braga, Portugal, mars 2007, volume 4422 de *LNCS*, pages 140-154. Springer.

[141] Jerry Tyan et Qusay H. Mahmoud : A network layer based architecture for service discovery in mobile ad hoc networks. *In Proceedings of the 17th Annual IEEE Canadian Conference on Electrical and Computer Engineering (CCECE 04)*, Niagara Falls, Canada, may 2004, volume 3, pages 1379-1384. IEEE CS.

[142] Unicore : Uniform Interface to Computing Resources. `http://www.unicore.eu`.

[143] UPnP Forum : UPnP Device Architecture, Version 1.1, octobre 2008. `http://www.upnp.org`.

[144] Rob van Ommering, Frank van der Linden, Jeff Kramer et Jeff Magee : The Koala Component Model for Consumer Electronics Software. *Computer*, 33(3):78-85, mars 2000.

[145] Alex Varshavsky, Bradley Reid et Eyal de Lara : A Cross Layer Approach to Service Discovery and Selection in Manets. *In Proceedings of the 2nd International Conference on Mobile Ad Hoc and Sensor Systems (MASS 05)*, Washington, DC, USA, novembre 2005. IEEE CS.

[146] Christopher N. Ververidis et George C. Polyzos : Service discovery for mobile ad hoc networks: A survey of issues and techniques. *IEEE Communications Surveys and Tutorials*, 10(3):30-45, 2008.

[147] W3C Consortium : SOAP Version 1.2. W3C Recommendation, avril 2007. `http://www.w3c.org`.

[148] W3C Consortium : Web Services Description Language (WSDL) Version 2.0. W3C Recommendation, juin 2007. `http://www.w3c.org`.

[149] Mark Weiser : The Computer for the 21st Century. *Scientific American*, 265(3):66-75, septembre 1991. Special Issue: Communications, Computers, and Networks.

[150] Bartosz Wietrzyk et Milena Radenkovic : Enabling Large Scale Ad Hoc Animal Welfare Monitoring. *In Proceedings of the International Conference on Wireless and Mobile Communications*, Cannes – La Bocca, France, août 2009, pages 401-409. IEEE CS.

[151] Zhensheng Zhang : Routing in Intermittently Connected Mobile Ad Hoc Networks and Delay Tolerant Networks: Overview and Challenges. *IEEE Communications Surveys and Tutorials*, 8(1):24-37, jan 2006.

[152] Zhensheng Zhang et Qian Zhang : Delay/disruption tolerant mobile ad hoc networks: latest developments. *Wireless Communications and Mobile Computing*, 7(10):1219-1232, mai 2009.

[153] Feng Zhu, Matt W. Mutka et Lionel M. Ni : Service Discovery in Pervasive Computing Environments. *IEEE Pervasive Computing*, 4(4):81-90, décembre 2005.

A

Encadrements doctoraux

Cette annexe décrit brièvement les encadrements doctoraux que j'ai effectués à l'Université de Bretagne-Sud depuis 2002.

Stage de DEA de Milad El Khodary (septembre 2002 –novembre 2002)

Milad El Khodary a effectué sous ma direction au Valoria son stage de DEA de l'Université de Beyrouth du 1er septembre au 29 novembre 2002. Ce travail, réalisé dans le cadre du projet Concerto, a porté sur le monitoring de ressources pour composants parallèles adaptables. Il s'agissait de définir un ensemble de modèles de ressources distribuées adaptés au contexte des grappes de stations de travail et de réaliser un prototype de moniteur pour ces ressources sur la grappe Myrinet disponible au Valoria.

Doctorat de Didier Hoareau (septembre 2003 – décembre 2007)

Didier Hoareau (titulaire d'un DEA de l'université de Nantes) a commencé sa thèse sous mon encadrement en septembre 2003 (allocation du ministère, puis poste d'ATER à partir d'octobre 2006). Cette thèse avait pour objectif de définir un modèle de déploiement et d'exécution pour les applications distribuées fondées sur l'utilisation de composants logiciels hiérarchiques. L'approche a consisté à enrichir un modèle de composants hiérarchiques tel que Fractal pour permettre la répartition des sous-composants sur une plate-forme distribuée. Un ensemble d'outils intergiciels ont été construits pour faciliter le déploiement de tels composants sur des réseaux de machines hétérogènes et volatiles tels que les réseaux d'équipements mobiles.

Ce travail de thèse a donné lieu à dix publications dans la période 2004–2008 dont un article dans une revue internationale [**j3**], et neuf publications dans des conférences internationales [**i8**, **i10**, **i7**, **i11**, **i9**, 69, 140, **i6**] et nationales [**n1**].

J'ai assuré l'encadrement scientifique de cette thèse. La direction administrative a été assurée par Patrice Frison, professeur au Valoria. Cette thèse a été soutenue le 5 décembre 2007 [68]. Le jury était composé de F. Oquendo, I. Demeure (rapporteur), L. Seinturier (rapporteur), P. Kuonen, P. Frison, Y. Mahéo. Didier Hoareau a obtenu un poste d'ATER à l'université de La Réunion, et après une activité en tant que chef de projet R&D dans la société ErgonHome (start-up réunionaise), il a été recruté comme maître de conférences à l'université de La Réunion (IUT de Saint Pierre) en septembre 2010.

Doctorat de Romeo Said (octobre 2006 – février 2011)

Romeo Said (titulaire d'un master recherche de l'université de Bretagne Occidentale à Brest) a commencé sa thèse sous mon encadrement en septembre 2006 (allocation régionale et départementale). L'objectif de la thèse était la construction d'un intergiciel de services pour les réseaux tolérant les délais. Il s'agissait de construire un ensemble d'outils logiciels facilitant le développement d'applications distribuées conçues comme une combinaison de services logiciels. Un intergiciel permettant la publication, la découverte et l'invocation de services devait être proposé. L'originalité de ces travaux réside dans le type des réseaux visés, dans la mesure où les services doivent être déployés sur des réseaux d'équipements mobiles et volatiles (de type ad hoc Wi-Fi) dans lesquels la connectivité de bout en bout n'est pas assurée. Ces travaux ont été effectués dans le cadre du projet ANR Sarah.

Ce travail de thèse a donné lieu à quatre publications dans des conférences et workshops internationaux [**i1**, **i4**, **i3**, **i2**].

J'ai assuré l'encadrement scientifique de cette thèse (à 70%), avec Frédéric Guidec, maître de conférences et directeur de la thèse. La thèse a été soutenue le 23 février 2011. Le jury était constitué de P.-F. Marteau, F. André (rapporteur), D. Donsez (rapporteur), T. Ledoux, F. Guidec, Y. Mahéo.

Doctorat d'Ali Makke (depuis octobre 2010)

Ali Makke (titualire du master Ubinet de l'université de Nice – Sophia Antipolis) a commencé sa thèse en octobre 2010 (allocation régionale et départementale). Le sujet de sa thèse porte sur la conception et la réalisation d'une plate-forme à services visant un environnement d'informatique ambiante hybride. cet environnement consiste en un réseau mobile ad hoc discontinu dans lequel sont présentes un certain nombre d'infostations fixes, pouvant jouer le rôle de relais entre le réseau mobile ad hoc et Internet. Les infostations sont reliées entre elles via Internet mais leur déploiement n'est pas planifié et leur nombre est insuffisant pour assurer une couverture radio de la zone considéré. Un protocole de communication adapté doit être conçu et développé, ainsi qu'un intergiciel assurant la découverte et l'invocation de services logiciels déployés dans le réseau d'infrastructure et accessibles via les infostations.

J'assure la direction de cette thèse qui est co-encadrée par Nicolas le Sommer, maître de conférences au Valoria.

B

Projets de recherche

Cette annexe présente, dans l'ordre chronologique, les différents projets dans lesquels se sont inscrits mes travaux de recherche depuis 2002.

Projet ACI Concerto

Le projet Concerto a été soumis dans le cadre de l'appel d'offre ACI GRID 2001 du Ministère de la recherche. Ce projet était porté initialement par le Valoria (L. Courtrai, F. Guidec et moi-même), l'Irisa (J.-L. Pazat et F. André) et le CEA (P. D'Anfray). Le Ministère à validé le projet dans la catégorie « Jeune équipe » sous réserve qu'il soit mené uniquement par le Valoria. Le projet Concerto a donc été reformaté et il a été financé à hauteur de 30 k€ pour la période 2002–2003. J'ai participé au montage de ce projet et j'en ai assuré la responsabilité.

L'objectif du projet Concerto était de contribuer à l'étude de l'application de la technologie à composants logiciels pour le Grid Computing, en proposant un modèle de composant parallèle adaptable. Dans cette optique, nous avons défini un modèle de construction des applications à base de composants et une infrastructure logicielle permettant à ces composants de percevoir l'environnement distribué dans lequel ils s'exécutent. L'originalité du modèle de composants Concerto réside dans le fait qu'il inclut la notion d'activité comme élément de base d'un composant et que les mécanismes permettant la perception de l'environnement sont pleinement intégrés au modèle. Un prototype Java implantant le modèle de composant Concerto a été developpé. Il est complété par un intergiciel qui permet d'une part la réification des ressources du système et de l'environnement applicatif et d'autre part l'observation dynamique de l'état de ces ressources. L'utilisation du prototype a permis la mise en œuvre d'un composant de démonstration constitué du code d'une application de routage actif développé au Laboratoire d'Informatique du Parallélisme (LIP, ENS Lyon).

Les travaux effectués durant le projet ont fait l'objet de 6 présentations dans des conférences (dont 4 d'audience internationale) entre 2002 et 2004, et d'un rapport de recherche présenté au Ministère [**i12**, **i13**, **i14**, **i15**, **n3**, **n2**, **r1**].

Projet Cubik

Le projet Cubik était un projet interne à l'équipe Casa du Valoria. Il s'est déroulé de 2003 à 2007 et les contributions principales ont été faites dans le cadre de la thèse de Didier Hoareau, sous ma direction. L'objectif du projet était de contribuer à l'étude de l'utilisation de composants logiciels pour le déploiement et l'exécution des applications distribuées qui visent des réseaux non systématiquement entièrement connectés. Le modèle de composants sur lequel nous nous étions appuyés était le modèle de composants Fractal, associé à son implantation de référence en Java Julia.

Les résultats principaux du projet Cubik ont consisté en la définition d'un modèle de composants hiérarchique adapté aux environnements ubiquitaires, la conception d'une méthode de déploiement originale fondée sur la prise en compte de contraintes, et la contruction d'un intergiciel permettant le déploiement de composants ubiquitaires et leur exécution distribuée.

Le projet Cubik a permis la publication de 9 articles dans des revues et conférences [**j3**, **i6**, 140, **i7**, 69, **i9**, **i8**, **i10**, **i11**].

Projet ANR Sarah

Le projet Sarah (Services Asynchrones pour Réseaux Ad Hoc) était un projet financé par l'Agence Nationale de la Recherche dans le cadre du programme ARA SSIA (Action de Recherche Amont, Sécurité, Systèmes embarqués & Intelligence Ambiante) pour la période allant de décembre 2005 à juin 2009. Ce projet, piloté par l'équipe Casa du Valoria, a impliqué trois autres laboratoires : LaBRI (Université de Bordeaux I), XLIM (Université de Limoges) et LITIS (Université du Havre). Le budget total du projet était de 272 k€.

Le projet SARAH s'est intéressé au déploiement et à l'exploitation de services distribués dans des réseaux tolérant les délais. Il vise plus particulièrement les réseaux mobiles ad hoc dans lesquels une connectivité de bout en bout ne peut être maintenue en permanence, notamment parce que le réseau peut être partitionné du fait de la volatilité ou de la forte mobilité des équipements. Un premier objectif était de définir et d'implanter un modèle de communication ad hoc asynchrone. Ce modèle a servi de base pour l'exploitation de services distribués asynchrones, c'est-à-dire de services pouvant être déployés, découverts et invoqués dans un environnement ad hoc partitionné, sur la base d'une combinaison de communication de proximité et de communication asynchrone. Une partie du projet a été consacrée aux problèmes de sécurité résultant de l'absence de connectivité de bout en bout dans le type de réseau considéré, en développant des solutions reposant sur l'emploi de cartes à puces et sur la dérivation de clés de chiffrement asymétriques à partir des identités des utilisateurs. D'autre part, des techniques de modélisation abstraite et de simulation ont été mises en œuvre en vue d'évaluer et d'améliorer le modèle de communication, et afin d'examiner comment des services asynchrones réalistes peuvent se comporter dans différents types de réseaux.

Le projet était structuré en quatre thèmes de recherche complémentaires :

1. Support de la communication dans un MANET discontinu.
2. Support au déploiement et à l'exécution de services dans ce type d'environnement.

3. Sécurisation des communications et des services.
4. Simulation et validation des mécanismes de communication et des services.

J'ai participé activement au montage du dossier déposé à l'ANR et assuré la responsabilité du deuxième thème couvert par le projet, thème qui impliquait le Valoria et le LaBRI. J'étais également responsable de la communication autour du projet à travers son site web (http://www-valoria.univ-ubs.fr/SARAH).

Les travaux effectués dans le projet SARAH ont fait l'objet de 37 publications dans des revues et conférences, dont 17 co-signées par un membre de l'équipe CASA [**j2**, 59, 92, **i1**, 60, 15, 14, **i2**, **i4**, **i3**, 56, 58, **i5**, 90, 89, 57, 91].