Conception et réalisation d'une plate-forme de répartition dédiée aux environnements nomades

Mejdi Kaddour

TELECOM PARIS
école nationale supérieure des télécommunications

Thèse

présentée pour obtenir le grade de docteur

de l'Ecole Nationale Supérieure des Télécommunications

Spécialité : Informatique et Réseaux

Mejdi Kaddour

Conception et Réalisation d'une Plate-forme de Répartition dédiée aux Environnements Nomades

soutenue le 14 mars 2005 devant le jury composé de

Pierre Sens	Université Paris 6	Président
Guy Bernard	Institut National des Télécommunications	Rapporteur
Laurence Duchien	Université Lille 1	Rapporteur
Gérard Vandome	Bull	Examinateur
Nicolas Rivierre	France Telecom R&D	Examinateur
Jean-Pierre Velu	Sagem	Invité
Laurent Pautet	Ecole Nationale Supérieure des Télécommunications	Directeur de thèse

à mes parents

Remerciements

Cette thèse est le fruit de trois années d'efforts incessants, mais aussi d'échanges bénéfiques et de collaborations fructueuses. Ce travail n'aurait pas pu aboutir sans le concours précieux et généreux de personnes qui partagent la même passion pour la recherche scientifique.

Je tiens à adresser, en premier lieu, mes remerciements les plus sincères à mon directeur de thèse, Monsieur Laurent Pautet, maître de conférence à l'ENST, pour la confiance qu'il m'a accordé tout au long de cette thèse. Je suis particulièrement reconnaissant envers ses avis éclairés ainsi qu'à l'implication continue dont il a fait preuve malgré ses nombreuses charges. J'ai beaucoup apprécié sa gentillesse et l'esprit de convivialité qui a animé tous nos rapports.

Je remercie naturellement, Monsieur Guy Bernard, professeur à l'Institut National des Télécommunications, et Madame Laurence Duchien, professeur à l'Université de Lille 1, de m'avoir fait l'honneur d'être les rapporteurs de cette thèse et de leurs remarques pertinentes concernant la version préliminaire de ce mémoire.

Je remercie vivement, Monsieur Pierre Sens, professeur à l'Université de Pierre et Marie Curie, d'avoir accepté de présider le jury de cette thèse.

Mes remerciements s'adressent également à Monsieur Gérard Vandome, responsable du projet JONAS à ObjectWeb, et Monsieur Nicolas Rivierre, expert senior à France Telecom R&D, d'avoir accepté de participer au jury.

Je remercie chaleureusement Monsieur Jean-Pierre Velu, expert auprès de la direction scientifique de Sagem, d'avoir gentiment accepté d'être mon invité d'honneur à la soutenance.

Je tiens à remercier Monsieur Michel Riguidel, chef du département INFRES à l'ENST, d'avoir mis en œuvre les moyens nécessaires à l'aboutissement de cette thèse. Je remercie également Madame Isabelle Demeure, professeur à l'ENST et responsable du groupe ASTRE au département INFRES, pour ses encouragements et pour l'intérêt qu'elle a toujours manifesté envers mon travail.

J'adresse mes remerciements à l'ensemble des participants au projet CARISM pour leur aide précieuse et leurs appréciations encourageantes. Je remercie également les stagiaires de DEA et les élèves de l'ENST qui ont participé à ce travail.

L'ensemble des permanents et des thésards du département Informatique et Réseaux de l'ENST sont priés de trouver ici l'expression de mes sincères sentiments d'amitié.

Je garde mes derniers remerciements pour ma famille. Aucune parole ne pourra jamais exprimer toute l'affection, la reconnaissance et l'estime que je ressens envers mes parents pour tout ce qu'ils m'ont apporté. Je pense aussi avec beaucoup d'émotion à mes deux défuntes grand-mères qui m'ont toujours porté par leur amour et leur bienveillance. Cet aboutissement est quelque part aussi le leur. Je remercie ma très chère soeur Siham et sa petite famille, mes tantes Fatiha et Houria, mon cousin Houari, pour leur soutien inconditionnel. Malgré la séparation et la distance, ils ont toujours été présents dans mon coeur.

Enfin, j'ai une pensée particulière pour mes amis d'Oran et d'ailleurs, pour leurs encouragements et pour les moments inoubliables que nous avons partagés.

Resumé

Les technologies intergicielles se révèlent de plus en plus comme des ingrédients indispensables dans les systèmes mobiles. Durant les dernières décennies, ces technologies ont connu une évolution considérable qui a permis leur implantation à grande échelle dans les environnements fixes traditionnels. Elles assurent des fonctions d'échange et d'interaction entre des applications hétérogènes hébergées par des machines différentes. Les intergiciels mobiles sont appelés à tenir le même rôle, ils doivent cependant prendre en considération des problématiques spécifiques aux environnements mobiles, comme la qualité variable du réseau ou les ressources limitées des machines.

Dans le cadre de cette thèse, nous avons mené une réflexion sur les différents types d'intergiciels et leurs aptitudes à relever le défi de la mobilité. Les différentes solutions existantes convergent vers un ensemble de propriétés et de recommandations communes comme l'asynchronisme, l'adaptabilité ou la configurabilité. Nous avons opté pour les intergiciels orientés messages (MOMs), car ils se caractérisent, par un style de communication permettant un fort découplage entre les entités réparties. La deuxième réflexion, d'ordre plus général, est que l'adaptabilité d'un système mobile ne peut être gérée efficacement qu'à travers un modèle global qui fait intervenir les utilisateurs, les applications, l'intergiciel et le contexte d'exécution.

Le premier volet de nos travaux concerne la conception et la réalisation de MobileJMS, un intergiciel basé sur la spécification *Java Message Service*. Par rapport aux MOMs existants, MobileJMS se distingue par l'adaptabilité et la sensibilité au contexte qui caractérisent la communication. MobileJMS intègre dans son module de communication des services qui peuvent être utilisés et configurés dynamiquement afin de réagir à des évènements comme la variation de la bande passante ou les déconnexions fréquentes. En particulier, le service de stockage et ré-émission propose une solution à la forte dépendance entre les applications et la connexion réseau. En outre, la structure de module de communication de MobileJMS permet de répondre à deux problématiques : l'accès transparent des applications à travers plusieurs réseaux et la gestion de la connexion intermittente (déconnexion/reconnexion).

Le deuxième volet concerne la définition et la mise œuvre d'un gestionnaire d'adaptation globale. Ce gestionnaire repose sur un modèle qui prend des décisions d'adaptation à partir des préférences des utilisateurs, des besoins des applications et des paramètres du contexte d'exécution. Nous exprimons les préférences des utilisateurs par des critères de performance comme la vitesse de transmission ou l'économie d'énergie. Les besoins des applications sont représentés sous forme de politiques d'adaptation. Une politique définit l'état du contexte d'exécution qui permet à une application de satisfaire des besoins donnés. La fonction principale du modèle est le choix de la politique la plus adaptée au contexte actuel et à l'usage que l'utilisateur souhaite faire de l'application (préférences). Nous avons, à ce propos, étudié l'impact de l'utilisation de la compression et de la fragmentation sur les critères de performance mentionnés. MobileJMS a fait l'objet d'une analyse des performances qui montre l'efficacité des décisions d'adaptation.

Abstract

Middleware technologies have emerged as a key element of future mobile systems. During the last decades, these technologies have seen a considerable evolution which allowed their large-scale establishment in traditional fixed environments. They ensure various functions like communication and interaction between heterogeneous applications located on different hosts. Mobile middleware systems should take the same responsibilities. However, they must consider some specific problems of mobile environments, like the variable quality of the network or the limited resources of mobile devices.

In this thesis, we reason about the various classes of middleware and their abilities to tackle various mobility challenges. The different existing solutions converge towards a whole set of properties and common recommendations like asynchronism, adaptability or configurability. We prefer the message oriented middleware systems (MOMs), because they are characterized by a communication style allowing a strong decoupling between distributed entities. The second thought, of a more general nature, is that the adaptability of a mobile system cannot be efficiently managed only through a global model which involve users, applications, middleware and execution context.

Our first contribution is related to the design and the implementation of MobileJMS, a middleware based on the Java Message Service specification. Compared to existing MOMs, MobileJMS is characterized by communication adaptability and context-awareness. The MobileJMS communication component is based on services which can be activated and configured dynamically in order to react to various events like bandwidth variation or frequent disconnections. In particular, the store and forward service proposes a solution to the strong dependence between applications and network connection. Moreover, the structure of this communication component cope with two major concerns : seamless application access through various networks and intermittent connection management (disconnection/reconnection).

Our second contribution consists of the definition and the implementation of a global adaptation manager. This manager is based on a model which makes adaptation decisions by considering user preferences, application requirements and execution context state. We state user preferences by performance criteria like transmission throughput or energy consumption. Application requirements are represented with adaptation policies ; a policy defines the execution context state enabling an application to meet some given requirements. The key function of the model is the selection of the most appropriate policy to the current context and the utilization the user has for the application. We have studied the impact of the use of compression and fragmentation on the mentioned performance criteria. MobileJMS was also subjected to a performance analysis which demonstrates the efficiency of adaptation decisions.

Table des matières

Table des figures

Liste des tableaux

Chapitre 1

Introduction

1.1 Contexte de la thèse

L'informatique mobile est un concept né des avancées technologiques spectaculaires dans le domaine des réseaux sans fil et de l'apparition de terminaux mobiles légers. Les technologies réseau actuelles comprennent des réseaux de télécommunications cellulaires, comme le GSM, le GPRS ou le très attendu UMTS, mais aussi des technologies destinées à un espace physique plus restreint comme le 802.11b ou Bluetooth. Les terminaux mobiles sont dotés de capacités de communication avec l'environnement extérieur et sont capables d'accompagner l'utilisateur dans tous ses déplacements. Ils prennent diverses formes et tailles allant du classique téléphone portable, jusqu'aux téléphones de troisième génération combinant les fonctions d'un assistant personnel et d'une interface de communication sans fil à un débit relativement important.

Les technologies et les services mobiles ont acquis au cours des dernières années une maturité qui a favorisé leur adoption à la fois par le grand public et par le monde professionnel. Ces technologies permettent à des utilisateurs d'accéder à des infrastructures partagées indépendamment de leur localisation physique. Pourtant, l'informatique mobile ne se réduit pas qu'à cela, elle constitue une véritable révolution dans la manière avec laquelle les utilisateurs emploient l'outil informatique, comme fût celle de l'apparition du Web. La machine de l'utilisateur n'est plus un espace confiné destiné à exécuter quelques applications, mais devient un véritable portail sur un espace d'applications et de données. Les utilisateurs, en se déplaçant à travers différents espaces, ne sont plus limités par des fonctionnalités disponibles au départ sur leur machine, mais auront la possibilité d'accéder dynamiquement à de nouvelles ressources, de charger de nouvelles applications et de partager de l'information.

Malgré son histoire récente, l'informatique mobile est basée sur un ensemble de concepts et de technologies relativement stables qui ont permis le développement d'applications innovantes. Cependant, les défis et les contraintes sont demeurés également relativement stables. Ces contraintes fondamentales, que Forman a bien cernées [26], sont inhérentes à la nature physique des environnements mobiles. Les terminaux mobiles possèdent beaucoup moins de ressources de traitement et de stockage que les machines fixes. Ils doivent communiquer dans un environnement incertain caractérisé notamment par une bande passante variable et des risques de déconnexion

non négligeables.

Ces défis ont amené les chercheurs à trouver des solutions dans des domaines différents, comme les réseaux, les systèmes d'exploitation, les systèmes répartis ou les bases de données. Les recherches dans les réseaux ont porté sur la conception de nouveaux mécanismes d'adressage, de routage, de diffusion (Multicast) et de gestion transparente de la mobilité (par exemple, Mobile IP). Les recherches dans les systèmes d'exploitation se sont focalisées sur les techniques de mise en cache et de conception de systèmes de fichiers prenant en charge les déconnexions. Dans le domaine des bases de données, les recherches ont abordé les problèmes de gestion de la réplication, de la concurrence et du traitement des transactions.

1.2 Intergiciel mobile : un élément fédérateur

Les systèmes répartis permettent l'interaction entre des applications évoluant sur des machines différentes d'un réseau de communication. Ils permettent également de fédérer et de mettre en commun les ressources de ces machines que se soit en terme de puissance de traitement ou en terme d'accès aux données. L'interaction entre les différentes entités s'effectue suivant un modèle de répartition bien précis (appel de procédure distante, passage de messages, mémoire partagée, etc.). Ce modèle est un ensemble de sémantiques qui décrivent ces interactions d'un point de vue logique plutôt qu'en termes des moyens nécessaires pour les mettre en œuvre.

Un intergiciel (*middleware*) peut être défini comme la concrétisation d'un modèle de répartition donné. Il fournit les abstractions et les mécanismes permettant aux applications d'interagir. Son rôle est de faciliter l'interaction en affranchissant les développeurs d'applications des problématiques liées à la répartition, comme le partage de ressources, le transport de données ou encore l'hétérogénéité des systèmes d'exploitation.

Les intergiciels mobiles représentent en quelque sorte la jonction entre les technologies mobiles et les systèmes répartis. Leur premier objectif est de mettre les abstractions de haut niveau et la richesse fonctionnelle des intergiciels au service des applications mobiles. Ce rôle est d'autant plus important que l'interaction est à la base des possibilités offertes par l'informatique mobile.

Les intergiciels mobiles ont également pour fonction de faire la passerelle entre le monde fixe et le monde mobile. Durant les dernières décennies, plusieurs technologies intergicielles ont été conçues et déployées sur les plates-formes fixes. Ces technologies sont devenues aujourd'hui indispensables dans beaucoup de domaines, comme l'intégration du patrimoine applicatif, souvent hétérogène, d'une entreprise. Elles permettent également de faciliter l'échange entre une entreprise et ses clients ou partenaires. Dans ce contexte, il existe un fort besoin d'augmenter la disponibilité des systèmes et des services et les rendre accessibles à des utilisateurs mobiles. Les intergiciels mobiles représentent un moyen, parfois incontournable, pour permettre à ces utilisateurs d'accéder et d'interagir avec des systèmes d'information dont les technologies intergicielles constituent la voie d'accès.

Compte tenu de la forte hétérogénéité des systèmes d'exploitation, des protocoles réseau, des types de données et des terminaux utilisés dans les environnements mobiles, les intergiciels mobiles doivent se comporter comme un élément fédérateur. Ils doivent prendre en considération les

problématiques de ces environnements, tout en offrant aux applications une interface homogène qui facilite les échanges entres diverses entités.

1.3 Approche et objectifs

L'objectif global de la thèse est de proposer et de mettre en œuvre un modèle d'adaptation permettant à un système mobile de réagir de manière appropriée et efficace au dynamisme qui caractérise les environnements mobiles. Ce modèle concerne plusieurs niveaux d'un système mobile : les utilisateurs, les applications, l'intergiciel et le contexte d'exécution.

Afin de pouvoir concrétiser ce modèle, il est nécessaire que le niveau intergiciel dispose de mécanismes d'adaptation. Les différentes technologies intergicielles existantes adoptent généralement un comportement rigide. Elles sont fondées sur l'hypothèse que l'environnement est stable et que les interactions se produisent entre des entités relativement homogènes en termes de ressources et d'accès réseau. Cette hypothèse est la plupart temps inappropriée, voire non valide, dans les environnements mobiles. Les intergiciels mobiles doivent fournir des mécanismes pour gérer des problématiques comme la mobilité, la variation de la bande passante, les déconnexions et l'introduction dynamique de nouvelles entités. Ils doivent être adaptables et sensibles à leur contexte d'exécution, contrairement à leurs homologues fixes qui adoptent généralement le principe de la transparence.

Nous avons conçu et réalisé l'intergiciel mobile, MobileJMS, qui appartient à la famille des intergiciels orientés messages (MOM). Il est basé sur la spécification JMS (*Java Message Service*). Nous voulons faire valoir l'idée, dans le cadre de cette thèse, que les MOMs constituent une solution efficace et intéressante aux problématiques des environnements mobiles. Les MOMs se caractérisent par un paradigme d'interaction asynchrone qui favorise les échanges entre des entités se connectant de manière intermittente et changeant fréquemment d'adresse physique sur le réseau.

La contribution principale de MobileJMS, hormis les avantages liés aux propriétés des MOMs, est l'adaptabilité et la sensibilité au contexte d'exécution qui caractérisent la communication entre les entités réparties. Cette adaptabilité se traduit par la capacité à réagir à des évènements comme la variation de la bande passante ou les déconnexions. En outre, MobileJMS propose une gestion de l'accès transparent à travers plusieurs réseaux.

MobileJMS fournit des outils pour réagir aux variations du contexte. Néanmoins, les décisions concernant la nature de l'adaptation à entreprendre dépendent d'autres niveaux du système. Nous avons conçu et mis en œuvre un modèle d'adaptation globale qui fait intervenir les préférences des utilisateurs et les besoins des applications. Ce modèle est basé sur la notion de politique d'adaptation qui définit une relation entre les besoins des applications et l'état du contexte d'exécution. Nous exprimons les préférences des utilisateurs par des critères de performance comme la vitesse de transmission ou l'économie d'énergie. Nous proposons donc un modèle qui permet de traduire un certain état du contexte en une configuration de l'intergiciel qui soit la plus adaptée à l'application et à l'usage que l'utilisateur souhaite en faire.

1.4 Cadre du travail

Cette thèse s'est déroulée au sein de l'équipe ASTRE du département Informatique et Réseaux de l'École Nationale Supérieure des Télécommunications. Le développement et l'exploitation de l'intergiciel MobileJMS a été effectué notamment dans le cadre de deux projets de recherche regroupant des partenaires des écoles des télécommunications (ENST, ENST-Bretagne, INT) : CARISM I et CARISM II (Composants Adaptables et Reconfigurables pour Intergiciels et Services Mobiles).

CARISM I avait pour objectif de définir et démontrer une plate-forme intergicielle dédiée aux réseaux ambiants et systèmes mobiles. Cette plate-forme comporte, notamment, les fonctions de découverte de services, de vérification de leur interopérabilité, et de reconfiguration de l'intergiciel pour la prise en compte de la qualité de service.

CARISM II vise à intégrer une notion d'adaptabilité globale et répartie de l'intergiciel et des services. L'infrastructure s'appuie sur l'intergiciel MobileJMS en introduisant de nouveaux services de haut niveau, tels que celui de dépôt de politiques d'adaptation. Une telle diversité de politique nécessite dans l'infrastructure une intelligence assurée par des techniques de vérification sémantique.

1.5 Plan du document

Le reste du document est organisé en deux parties.

Etat de l'art La première partie aborde les différentes problématiques concernant les environnements mobiles et la conception des intergiciels mobiles.

Le chapitre 2 constitue une introduction à deux cas particuliers des environnements mobiles : les environnements nomades et les environnements diffus. Cette introduction aborde ces environnements du point de vue des défis qu'ils dressent. Une partie de ce chapitre est consacrée au concept de l'adaptabilité en citant quelques techniques qui concernent différents aspects des systèmes mobiles.

Le chapitre 3 présente les intergiciels classiques destinés aux environnements fixes. Nous étudions en particulier les MOMs et la spécification JMS. En outre, nous décrivons l'intergiciel générique Jonathan qui offre les briques de base dans la construction de MobileJMS.

Le chapitre 4 fait un tour d'horizon des différents types d'intergiciels mobiles. Chaque type d'intergiciel est illustré par des exemples. Nous abordons également les services essentiels proposés par ces intergiciels, comme la découverte de services ou le partage de données. Ce chapitre se conclut par une discussion qui dégage les propriétés nécessaires aux intergiciels mobiles.

Conception et Réalisation Cette deuxième partie concerne la conception et le développement de MobileJMS ainsi que du modèle d'adaptation proposé.

Le chapitre 5 présente l'approche et l'architecture de MobileJMS. Nous décrivons notamment le module de communication et ses propriétés d'adaptation et de configuration. Ces propriétés sont assurées à travers des services comme la compression et la fragmentation des données. Nous détaillons également la conception et la mise en œuvre de l'accès multi-réseaux.

Le chapitre 6 définit le modèle d'adaptation. Une étude est consacrée à la relation entre la compression et la fragmentation, et les critères de performance des applications. Cette étude nous permet de déduire des règles concernant la mise en œuvre des décisions d'adaptation.

Le chapitre 7 concerne la vérification formelle de deux mécanismes utilisés dans MobileJMS : la négociation et la reconnexion transparente. Cette vérification est nécessaire vu le nombre important d'états que ces deux mécanismes sont susceptibles de générer.

Nous avons effectué des mesures de performance sur MobileJMS. Nous en discutons les résultats dans le chapitre 8. Nous présentons également une application témoin qui illustre le fonctionnement de MobileJMS et du modèle d'adaptation.

Enfin, le chapitre 9 conclut ce document en synthétisant les contributions essentielles de notre travail et en évoquant les perspectives qu'il ouvre.

Première partie

Etat de l'Art

Chapitre 2

L'informatique nomade et diffuse

Les possibilités offertes par les technologies mobiles ont donné naissance à deux paradigmes : l'informatique nomade et l'informatique diffuse. L'un des principaux objectifs de l'informatique nomade est de fournir aux utilisateurs mobiles un accès sans fil aux systèmes qui existent sur les infrastructures fixes, comme l'illustre l'intégration des technologies cellulaires avec le Web. Les utilisateurs peuvent accéder au système d'information de leur entreprise alors qu'ils sont loin de leur poste de travail.

L'informatique diffuse ou omniprésente (*pervasive, ubiquitous*) peut être définie simplement comme l'enfouissement et la mise à disposition de l'informatique de manière globale, libre et intuitive. L'informatique diffuse propose également de démanteler les barrières qui existent entre les différents instruments de notre vie quotidienne. Ces instruments ne sont pas seulement les terminaux personnels auxquels on pense d'habitude, mais des équipements ou des dispositifs très miniaturisés et parfois même invisibles. Cet ensemble inclut tous les types d'objets imaginables, qu'ils soient mobiles ou intégrés : voitures, réfrigérateurs, vêtements, capteurs et différents biens de consommation. Ces différents instruments devront interagir et coopérer afin d'améliorer le service fourni et la satisfaction à l'utilisateur. Dans un scénario idéal, l'informatique diffuse doit se glisser dans chaque terminal ou équipement possédant une capacité de traitement. Selon Dan Russell (IBM)[88], « en 2010 l'informatique deviendra si naturelle dans l'environnement, que les gens ne réaliseront même plus qu'ils sont en train d'utiliser des ordinateurs ».

Ce chapitre est une introduction aux problématiques de chacun de ces deux paradigmes, ainsi qu'aux tendances actuelles visant à les résoudre. Cette présentation concerne davantage leurs aspects logiciels et applicatifs. Nous nous intéressons également au concept d'adaptabilité qui s'est révélé comme fondamental dans les environnements nomades et diffus, et qui nous concerne tout particulièrement dans le cadre de cette thèse. Nous citons quelques-unes des techniques d'adaptation existantes.

2.1 L'informatique nomade

Les environnements nomades sont généralement composés d'une partie réseau sans fil et d'une partie fixe. La partie sans fil peut être formée en utilisant deux approches. La première approche consiste à utiliser les réseaux de télécommunications cellulaires pour fournir un accès sans fil aux stations fixes. L'avantage de cette approche est la large couverture physique et l'intégration de mécanismes pour la gestion du déplacement des utilisateurs d'une cellule vers une autre (*handoff*). La deuxième approche consiste à utiliser les réseaux locaux sans fil, comme les technologies 802.11b [30], Bluetooth [10], HomeRF [57] ou HiperLan2 [40]. Ces réseaux, bien que ne pouvant être déployés que sur un périmètre physique restreint, fournissent une bande passante plus importante que celle des réseaux cellulaires.

2.1.1 Les caractéristiques des environnements nomades

Les environnements nomades se distinguent des environnements fixes par plusieurs aspects importants.

L'accès sans fil Contrairement aux utilisateurs fixes, les utilisateurs nomades se connectent à travers des réseaux sans fil. Ces réseaux présentent un environnement plus contraignant que les réseaux fixes : bande passante limitée et variable, plus de latence, risque de déconnexion important et un taux d'erreurs plus élevé. De plus, le lien sans fil n'offre pas les mêmes garanties en termes de fiabilité et de disponibilité. Ces contraintes font que la performance des applications et du réseau soit l'une des préoccupations majeures dans les environnements nomades. Parmi les paramètres de performance influents : le débit offert aux applications, le temps de réponse, la latence et le temps d'initialisation des connexions. Par exemple, le protocole TCP s'est avèré moins efficace dans ces environnements, en raison du long échange de paquets de contrôle lors de l'initialisation des sessions ou de la dégradation significative de la bande passante utilisée en cas de perte de paquets [13].

La mobilité des utilisateurs La mobilité des utilisateurs présente plusieurs défis. Les utilisateurs peuvent se retrouver hors de la zone de couverture du réseau lors de leurs déplacements. Les connexions peuvent être perdues ou dégradées lors du passage d'une cellule à une autre. Le routage des données doit tenir compte de la localisation des utilisateurs, car l'adresse d'un utilisateur nomade n'indique pas forcément sa localisation. En conséquence, l'infrastructure sous-jacente nécessite des procédures pour suivre et localiser les utilisateurs à tout moment. Ces procédures existent dans les réseaux cellulaires qui utilisent des bases de données pour mettre à jour la localisation des utilisateurs.

Les limites des terminaux mobiles Les utilisateurs nomades sont dotés généralement de terminaux légers, comme les assistants personnels ou les téléphones cellulaires. Ces terminaux offrent moins de capacités et de ressources que les machines fixes de bureau. Ils possèdent moins de puissance de calcul, moins de capacité mémoire et moins de capacité de stockage. En outre,

leur autonomie en énergie, bien qu'elle soit régulièrement en augmentation, reste limitée et réduit leur performance et leur convivialité.

2.1.2 Les problématiques des environnements nomades

Les caractéristiques, que nous avons soulignées ci-dessus, impliquent une prise en charge nouvelle ou différente des problématiques connues des environnements répartis fixes. Nous présentons ci-après, de manière non exhaustive, quelques-unes de ces problématiques.

Le mode d'accès et le partage des données

L'accès mobile aux données doit assurer le chargement des données à partir d'un serveur, qu'il soit fixe ou mobile, et le maintien de la cohérence entre les données du client et du serveur. L'accès aux données peut être initié par le serveur (*server push*) ou initié par le client qui envoie des requêtes au serveur pour obtenir les informations (*client pull*). Il peut être aussi un compromis entre les deux (hybride) [49]. Dans la plupart des applications (par exemple, le Web) le volume de données transmis du serveur vers le client est beaucoup plus important que celui qui est transmis dans le sens inverse. Dans ce type de communication asymétrique, si les données transmises par le serveur s'adressent à un nombre important de clients, il est plus approprié d'utiliser un mode de diffusion du serveur vers les clients (vidéo à la demande). Des techniques de multiplexage ont été utilisées pour envoyer plusieurs flux sur un même canal [46]. La fréquence de transmission d'un flux par rapport aux autres flux est déterminée par le nombre de destinataires.

Par ailleurs, la nature des réseaux sans fil impose la réduction au maximum du degré de couplage entre les activités du client et du serveur. Les terminaux doivent pouvoir accomplir des traitements même en cas d'absence de connexion avec le serveur. Par conséquent, un schéma de réplication est nécessaire pour garder une copie locale de chaque objet partagé. La réplication introduit le problème de la cohérence entre les différentes copies qui évoluent indépendamment. Des mécanismes de synchronisation sont utilisés pour prévenir le problème ou pour résorber les incohérences (voir section 2.3.3).

La mise en cache est une technique qui peut s'avérer très efficace dans le cas d'accès répétés à certaines données. Le pré-chargement (*prefetching*) [50] consiste à récupérer localement les données avant la déconnexion. La difficulté liée à cette approche est le choix des données. Les solutions possibles sont de ne retenir que les données accédées le plus récemment ou de solliciter l'utilisateur dans la sélection de ces données. La première approche suppose que les données accédées soient toujours les mêmes, alors que la deuxième nécessite une expertise et une implication que l'utilisateur n'est pas toujours disposé à fournir. Certains systèmes adoptent une approche prédictive basée sur le concept de relations sémantiques entre les données [56]. Un composant est chargé d'observer le comportement de l'utilisateur et de récolter les informations qui servent à faire des corrélations entre les données en calculant des distances sémantiques. Les données sont ainsi regroupées dans des classes de sorte qu'il existe une forte probabilité que celles requises par le client se trouvent dans une seule classe.

La gestion de la cohérence dans les architectures client/serveur classiques est généralement

classée en deux catégories :

1. L'approche « rappel de service »(*callback*) : les serveurs envoient des messages d'invalidation aux clients ayant mis dans leur cache des objets qui ont été modifiés.
2. L'approche « détection »(*polling*) : les clients envoient des requêtes vers le serveur pour valider leur cache.

La difficulté de l'application de ces deux approches dans les environnements nomades réside dans les déconnexions fréquentes et la faible bande passante qui entraînent parfois des délais importants. Une solution à ce problème consiste, pour les clients, à vérifier l'état du serveur à différents niveaux de granularité afin d'éviter d'effectuer systématiquement des validations pour chaque objet [69]. Une autre solution consiste à utiliser des canaux de diffusion. Le serveur envoie périodiquement des rapports d'invalidation sur les objets qui ont été modifiés. Les clients « écoutent » ces canaux en attendant un rapport les concernant. Cette approche offre l'avantage que le serveur ne doit pas nécessairement connaître l'adresse et l'état de tous ses clients.

Partage dynamique des tâches entre le client et le serveur

Les disparités qui existent entre les performances des terminaux mobiles et des machines fixes, suggèrent de tirer profit des capacités de ces dernières à chaque fois que cela est possible. L'exécution d'une tâche sur un serveur est généralement plus avantageuse lorsque la qualité du réseau est bonne. Par exemple, une page de mémoire peut être stockée dans la mémoire d'un serveur au lieu d'être stockée localement sur le disque du terminal afin d'économiser l'énergie. Par contre, dans un environnement caractérisé par des déconnexions fréquentes et une bande passante très variable, il serait plus approprié d'exécuter la tâche localement.

La plate-forme Rover [50] propose, dans ce sens, une alternative au modèle client/serveur classique, en permettant au client et au serveur de continuer d'effectuer les traitements même en cas de déconnexion. L'objet sur lequel s'effectue la requête est copié dynamiquement du serveur vers le client qui le stocke dans un cache local. Toutes les requêtes ultérieures sur cet objet s'opèrent localement, ce qui réduit la dépendance du client vis-à-vis du réseau. Bien entendu, les modifications qui peuvent intervenir sur la copie sont propagées vers l'objet primaire lors de la reconnexion. Ce modèle doit cependant être accompagné de mécanismes qui permettent de préserver la cohérence entre l'objet primaire et ses copies locales. Parmi ces mécanismes, citons le verrouillage, qui constitue l'approche pessimiste préservant une forte cohérence au détriment de la performance, et les algorithmes de réconciliation de données qui constituent l'approche optimiste.

Optimisation de la taille du flux de données

Les mécanismes, qui permettent de minimiser le flux de données échangées sur le réseau, trouvent dans les réseaux sans fil un vaste champ d'application. Parmi lesquels :

– La compression : éliminer les informations redondantes du flux de données.

- Le filtrage : ne transmettre que les données utiles.
- Les mises à jour différenciées : ne transmettre que les informations nécessaires pour synchroniser les données de l'émetteur et du récepteur.

Néanmoins, en sachant que des mécanismes, comme la compression et la décompression, requièrent un nombre important de calculs qui entraînent une consommation excessive de l'énergie, leur emploi sur les terminaux mobiles n'est pas toujours approprié. Toute la difficulté consiste donc à trouver un compromis entre les capacités du réseau et les capacités de traitement et d'économie d'énergie des terminaux.

Tolérance aux fautes

Les risques de défaillance sont considérables dans les environnements nomades. Les techniques de tolérance aux fautes doivent, par conséquent, être plus rigoureuses. Plusieurs techniques de détection de fautes et de reprise de panne peuvent être employées, mais elles doivent être adaptées pour prendre en considération les spécificités de ces environnements. Parmi lesquels :

- Adapter les techniques de point de reprise : dans les environnements mobiles, le terminal peut être un portail vers une application qui s'exécute à distance. Ce modèle applicatif doit être réexaminé dans ce contexte, parce que le type et la fréquence des points de reprise peuvent être différents de ce qu'ils sont dans l'état actuel de l'art.
- Détecter les déconnexions : les systèmes répartis traditionnels ont du mal à faire la distinction entre la défaillance du système distant et la déconnexion. Cette distinction est un facteur clé, dans ce contexte, car la déconnexion est un événement qui fait partie intégrante du système et doit être traitée différemment de la défaillance.

Interopérabilité

La question de faire coopérer et communiquer différents composants logiciels sur les réseaux fixes, a été largement présente ces dernières années [105]. Les environnements nomades doivent bénéficier plus que d'autres de ces efforts, mais ils introduisent en même temps une difficulté supplémentaire qui est la recherche de la performance. Les plates-formes CORBA [79], par exemple, offrent un large éventail de services qui enrichit les aspects non fonctionnels des applications réparties. Mais leur généralisation se heurte parfois à leur coût en matière de performance sur les environnements contraignants. Toute la difficulté du problème de l'interopérabilité réside dans la nécessite de faire un compromis entre deux objectifs opposés : d'un côté, la conception d'un ensemble englobant des technologies différentes ; de l'autre, le respect des contraintes des environnements nomades.

Sécurité et confidentialité

Du fait de la facilité de l'accès physique à un réseau sans fil, la sécurité est plus vulnérable que sur un réseau fixe. Les problèmes classiques sont le piratage, l'usurpation d'identité ou le

déni de service. La sécurité est un problème complexe qui doit être géré à tous les niveaux en incluant des mécanismes comme le cryptage ou l'authentification. Les mécanismes de sécurité doivent inclure également le fait que l'utilisateur mobile soit par définition autorisé à accéder à d'autres domaines.

Résumé

Les problématiques des environnements nomades tiennent essentiellement aux ressources limitées des terminaux et à la qualité variable de la connexion réseau. Les solutions présentées dans cette section sont basées sur un ensemble d'idées communes.

- Réduire la dépendance des clients mobiles vis à vis du réseau (pré-chargement, réplication, téléchargement des objets distants)
- Transférer le traitement des tâches vers le serveur lorsque les ressources des clients sont limitées.
- Optimiser l'usage de la bande passante réseau (multiplexage, diffusion (*broadcasting*), réduction de la taille du flux de données).
- Adapter le mode d'interaction entre le client et le serveur (invocation locale, rappel de service, détection).

2.2 L'informatique diffuse

L'informatique diffuse est un concept qui recouvre celui de l'informatique nomade. Le slogan de l'informatique nomade, qui est « n'importe quel moment, n'importe où »(*anytime anywhere*), se voit attribuer une dimension supérieure avec l'informatique diffuse : « tout le temps, partout »(*all the time everywhere*). Mais si la nature des environnements nomades est adaptée aux réseaux cellulaires, la nature des environnements diffus est plutôt adaptée aux réseaux sans fil ad hoc [86]. Ces réseaux sont constitués spontanément par l'interaction de deux ou de plusieurs machines et ne nécessitent pas la présence ou l'accès à une infrastructure fixe. Cette relation basée sur la proximité physique est également présente dans les systèmes diffus qui exploitent les ressources et les liens de communication disponibles dans le voisinage.

L'informatique diffuse est plus centrée sur l'environnement que l'informatique répartie fixe ou nomade. Cela signifie que les applications et le système se voient affectés des responsabilités encore plus larges. Des capteurs spécialisés sont utilisés pour recueillir automatiquement quelques informations représentant l'environnement de l'utilisateur. Ces informations permettent de construire une représentation partielle ou totale de l'environnement. Elles sont traduites dans un format qui peut être exploité par le système ou par les applications.

Parmi les différentes technologies diffuses qui émergent actuellement, citons : les ordinateurs « vestimentaires »(*wearable computers*) [22] et les maisons ou les immeubles intelligents. Une maison intelligente est une maison équipée d'une structure qui permet à ses occupants de programmer ou de contrôler à distance un ensemble de dispositifs électroniques. Ces commandes peuvent être déclenchées sur l'initiative de l'utilisateur d'une manière automatique lorsque cer-

tains évènements se produisent. Par exemple, l'utilisateur peut employer son téléphone portable pour activer le système d'alarme ou pour contrôler le niveau de température et de luminosité. Le niveau de luminosité peut aussi être géré automatiquement par le système en ajustant les stores ou en modulant la lumière électrique.

En plus des problématiques des environnements nomades, que nous avons citées précédemment, les systèmes diffus doivent faire face à des défis encore plus importants.

Hétérogénéité et Intégration

L'informatique diffuse se caractérise par l'extrême hétérogénéité des plates-formes matérielles et des technologies de communication. En effet, les dispositifs employés peuvent varier d'un capteur miniaturisé enfoui dans l'environnement à une machine puissante chargé du traitement de gros volumes de données. Ces dispositifs possèdent des caractéristiques très différentes concernant la puissance de traitement et de stockage ainsi que les possibilités de communication. Les développeurs ont dû résoudre, par exemple, d'importants problèmes qui résultent des incompatibilités entre les protocoles réseau.

Mais à notre avis, la difficulté principale se concentre au niveau des applications. Aujourd'hui, les applications sont développées pour des classes spécifiques de plates-formes ou de systèmes, ce qui entraîne souvent le développement de versions différentes pour les terminaux légers, les postes de travail ou les gros serveurs. L'une des pistes à explorer pour réduire cette complexité reste l'utilisation de plates-formes ou de langages relativement portables et neutres comme la plate-forme Java.

L'intégration est un problème qui est lié à cette hétérogénéité. Les environnements diffus possèdent généralement des architectures très complexes, formées de composants ayant des considérations différentes en matière de qualité de service, de fiabilité et de sécurité. Le besoin de coordination entre ces composants est donc clair. Par exemple, le format de données utilisé pour la communication entre les serveurs, n'est souvent pas exploitable sur les terminaux légers en raison de sa lourdeur et de sa complexité.

Passage à l'échelle

Les environnements diffus se caractérisent par la multiplication des utilisateurs, des applications et des terminaux utilisés. Plus il y a de l'intelligence dans ces environnements, plus le nombre d'interactions homme-machine et machine-machine croît. Ces interactions se déroulent, il faut le rappeler, dans un contexte introduisant des limites parfois draconiennes en ce qui concerne la puissance de traitement et le volume des données qui peut être transporté.

Dans ce contexte, le passage à l'échelle nécessite également un effort considérable de déploiement et de maintenance des applications. La convergence des plates-formes peut cependant atténuer cet effort en uniformisant la gestion des classes différentes de matériel.

Autonomie

Un système devient de plus en plus autonome en minimisant le nombre d'interventions humaines nécessaires. Ces interventions doivent se réduire aux cas où les environnements intelligents ne peuvent automatiquement prendre en charge les besoins des utilisateurs. Par exemple, un système de gestion domestique doit pouvoir identifier automatiquement la personne qui pénètre dans une pièce, et régler l'ambiance (température, luminosité, etc.) selon ses souhaits. Un utilisateur mobile peut imprimer un document sur l'imprimante la plus proche sans avoir recours à un service centralisé.

Nommage

Dans les environnements nomades, le terminal mobile, qui arrive dans un nouveau réseau, doit disposer d'un nom afin qu'il puisse communiquer avec les services locaux. Ce nom lui est attribué par une autorité centrale, il lui permet de pourvoir recevoir les messages qui lui étaient destinés sur l'ancien réseau. Le nom est donc associé dynamiquement avec l'adresse réseau actuelle du terminal. Cette démarche n'est pas toujours applicable dans les environnements diffus où l'accès à une autorité centrale de nommage, comme le DNS, n'est pas garanti.

Découverte de ressources

L'un des principaux avantages des environnements diffus est de pouvoir de bénéficier des ressources disponibles dans le voisinage physique (services, stockage, information, etc.). La découverte des ressources matérielles est généralement assurée au niveau des couches basses du système. La norme Bluetooth [10] fournit, par exemple, un mécanisme permettant de détecter les autres appareils Bluetooth présents dans le voisinage. Lors d'un déplacement, le terminal peut changer de domaine. Les performances des services qui étaient assurés dans son domaine d'origine peuvent se dégrader car le chemin réseau utilisé est alors plus long, ce qui introduit plus de latence et de risques de déconnexion. Le terminal doit donc pouvoir accéder aux services locaux et cela de manière transparente, sans nécessiter une intervention humaine. Des protocoles sont donc nécessaires au terminal pour détecter les services accessibles, pour partager des informations de configuration avec eux et pour être notifié quand ils ne sont plus disponibles. La détection doit être périodique et efficace afin de ne pas inonder le réseau.

Résumé

Les défis des environnements diffus vont demeurer un terrain fertile aux travaux de recherche dans les prochaines années. Du fait de l'absence d'entité centralisée et du caractère très dynamique de la connexion, les environnements diffus posent des difficultés supplémentaires par rapport aux environnements nomades. Les solutions présentées dans la section précédente doivent être repensées car la distinction entre le contexte d'exécution et les capacités des clients et des serveurs n'est plus aussi évidente. Par exemple, le nommage et la découverte de ressources dans

les environnements nomades adoptent un modèle centralisé qui n'est pas applicable dans les environnements diffus. En outre, contrairement aux environnements nomades, les possibilités de passage à l'échelle ne doivent pas se baser sur la présence d'entités pouvant accomplir la plus grande partie du traitement.

2.3 L'adaptation dans les environnements mobiles

La variabilité et les limitations des environnements mobiles, qu'ils soient nomades ou diffus, font de l'adaptation un mécanisme fondamental pour la conception d'applications mobiles. L'adaptation englobe toutes les stratégies et les techniques mises en œuvre par le système et les applications afin de répondre aux variations du contexte d'exécution. Ces variations peuvent se produirent à différents niveaux : la bande passante réseau, les réseaux disponibles, la localisation géographique ou les ressources disponibles sur le terminal.

Les techniques d'adaptation ont été classées dans un intervalle délimité par deux extrêmes [49]. D'un côté, l'adaptation est entièrement de la responsabilité des applications. De l'autre côté, l'adaptation est entièrement de la responsabilité du système. Dans ce cas, l'adaptation est transparente à l'application. Un cas typique de cette approche est l'utilisation d'agents pour le compte des applications [41][16]. Entre ces deux extrêmes réside un large spectre d'approches qui nécessitent une collaboration entre le système et les applications. Les applications ont une meilleure connaissance de ce qu'il faut faire en ce qui les concerne, lors d'un changement dans le contexte. Le système se charge généralement de la supervision et l'allocation des ressources. Il faut souligner cependant que les développeurs d'applications ne peuvent pas prévoir au départ toutes les situations possibles. Le système doit réserver également la possibilité d'introduire de manière dynamique de nouveaux comportements. La liaison de bibliothèques dynamiques ou la réflexivité (voir section 4.2.1) sont des exemples de mécanismes qui peuvent être employés dans ce cadre.

L'adaptabilité peut donc s'insérer à différents niveaux et peut prendre plusieurs formes. Nous présentons ci-après quelques techniques d'adaptation qui concernent différents aspects : application, système et représentation des données. Cette présentation concerne davantage les aspects liés à cette thèse. Notons que d'autres techniques d'adaptation sont citées dans le chapitre consacré aux intergiciels mobiles.

2.3.1 Adaptation du contenu

La plupart des informations, échangées quotidiennement par des millions d'utilisateurs et d'applications à travers le monde, sont représentées avec des formats conçus pour des environnements fixes. L'exemple le plus connu est sûrement le Web. Les navigateurs Internet permettent aux utilisateurs d'explorer l'immense masse d'information éparpillée sur la toile. Le langage et le protocole communément utilisés pour formater et présenter le contenu de l'information, et pour transférer les données entre un serveur et un navigateur Web, sont respectivement, HTML (*HyperText Markup Language*) et HTTP (*HyperText Transfer Protocol*). Ces deux outils sont conçus pour opérer dans des environnements caractérisés par des ressources machine importantes et une bande passante réseau confortable. Ces conditions ne sont pas toujours satisfaites

dans les environnements mobiles, l'adaptation du format et du contenu des informations devient donc nécessaire.

L'un des objectifs de l'adaptation du contenu est de permettre aux utilisateurs d'accéder à l'information en utilisant tout type de terminal et tout type d'affichage sur l'écran. Ce type d'adaptation est possible grâce notamment à la diversité des formats de données disponibles actuellement. Chaque terminal doit posséder un profil qui contient des informations à propos de ses capacités. Ce profil est utilisé pour déterminer le format de données qui est adapté au terminal de l'utilisateur. L'adaptation du contenu implique la conversion d'un format de données vers un autre, comme de HTML à XHTML [111] ou WML [106]. Nous décrivons ci-après le standard CC/PP qui apporte une contribution dans ce sens.

Le langage de description CC/PP (*Composite Capability/ Preference Profile*)

CC/PP [112] fournit les informations nécessaires à la sélection du contenu et du mécanisme de transport qui correspondent le plus aux capacités du terminal et aux préférences de l'utilisateur. Ces informations sont stockées sous forme d'un profil de terminal. CC/PP est basé sur le langage RDF [113] qui utilise la notion de meta données (des données à propos de données) pour décrire les ressources du Web et pour permettre aux applications d'échanger des informations sous une forme compréhensible par les machines. RDF définit un modèle pour décrire les ressources tout en n'émettant aucune hypothèse sur un domaine applicatif particulier. Le modèle RDF de base consiste en trois types d'objets :

1. **Ressources :** tout objet décrit par RDF est appelé une ressource. Une ressource peut représenter une page Web ou un site Web tout entier. Les ressources sont toujours désignées par des URIs (*Uniform Resource Identifier*).
2. **Propriétés :** une propriété est un aspect, une caractéristique, un attribut ou une relation, spécifiques, qui est utilisée pour décrire une ressource. Chaque propriété a sa signification particulière. Elle définit les valeurs qu'elle autorise, les types des ressources qu'elle peut décrire et les relations avec les autres propriétés.
3. **Déclarations :** une ressource spécifique, une propriété nommée, ainsi qu'une valeur de cette ressource, forment une déclaration. Ces trois parties de la déclaration sont appelées, respectivement, le sujet, le prédicat et l'objet.

Un profil CC/PP est une description des capacités d'un terminal et des préférences d'un utilisateur. Ci-dessous, un exemple d'un profil CC/PP décrivant les capacités d'un terminal mobile et les préférences de l'utilisateur.

```
<?xml version="1.0"?>
<rdf:RDF
   xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
   xmlns:prf="http://www.w3.org/TR/WD-profile-vocabulary#">
     <rdf:Description about="HardwarePlatform">
        <prf:Defaults
            Vendor="Nokia"
            Model="2160"
```

```
            Type="PDA"
            ScreenSize="800x600x24"
            CPU="PPC"
            Keyboard="Yes"
            Memory="16mB"
            Bluetooth="YES"
            Speaker="Yes" />
         <prf:Modifications
            Memory="32mB" />
      </rdf:Description>
      <rdf:Description about="SoftwarePlatform">
         <prf:Defaults
            OS="EPOC1.0"
            HTMLVersion="4.0"
            JavaScriptVersion="4.0"
            WAPVersion="1.0"
            WMLScript="1.0" />
         <prf:Modifications
            Sound="Off"
            Images="Off" />
       </rdf:Description>
       <rdf:Description about="EpocEmail1.0">
          <prf:Defaults
             HTMLVersion="4.0" />
          </rdf:Description>
          <rdf:Description about="EpocCalendar1.0">
             <prf:Defaults
                HTMLVersion="4.0" />
       </rdf:Description>
       <rdf:Description about="UserPreferences">
           <prf:Defaults
                Language="English"/>
       </rdf:Description>
</rdf:RDF>
```

Un serveur compatible CC/PP doit utiliser les profils pour adapter le contenu des données envoyées par rapport au terminal du client destinataire. Le serveur doit donc avoir accès au profil du client afin de négocier le contenu. Il peut accéder à ce profil, soit en demandant au client d'envoyer le profil complet, soit en récupérant le profil par défaut. Celui-ci est établi par le constructeur du terminal, sa localisation est décrite par une URI.

En plus du profil du terminal, le serveur utilise un profil de document qui spécifie les ressources nécessaires pour lire le document. Par exemple, dans le cas d'une page Web, le profil de document décrit les outils nécessaires pour pouvoir l'interpréter (XHTML, XML, CSS, etc.). Le serveur effectue une correspondance entre le profil du document et le profil du terminal afin de générer le meilleur contenu possible pour le terminal, comme l'explique la figure 2.1.

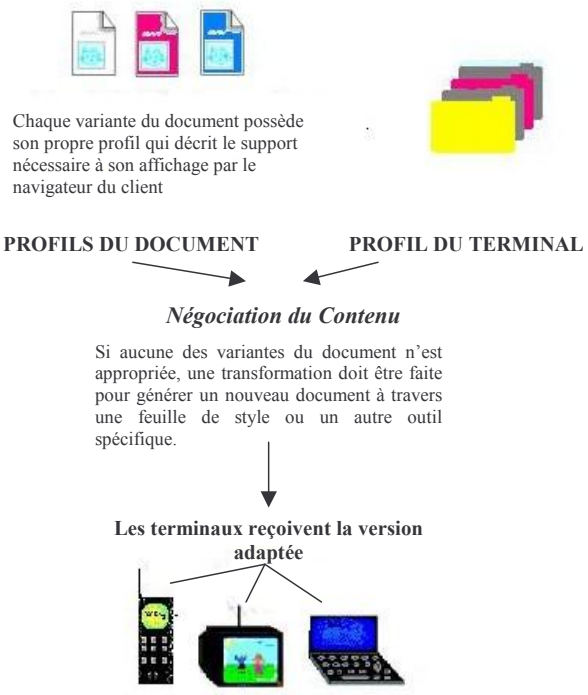


Fig. 2.1 – Adaptation basée sur CC/PP

2.3.2 Adaptation des données

Les données peuvent être adaptées en utilisant des techniques qui dépendent de leurs types. Ces techniques ont pour objectif d'améliorer les performances, comme la compression de l'en-tête du paquet TCP dans les réseaux bas débit. Les données de haut niveau sont plus complexes et nécessitent des techniques différentes pour les adapter aux variations des conditions réseau. Ces techniques reposent généralement sur un mandataire (*proxy*) qui effectue ces transformations pour le compte du client qui reçoit les données. Parmi les techniques d'adaptation de données :

- **La compression générique :** la compression préserve généralement le contenu des données échangées mais opère un changement de leur structure impliquant la diminution de leur taille.

- **La dégradation :** la dégradation préserve généralement la structure des données échangées mais le contenu des données reçues n'est qu'un sous-ensemble des données envoyées. Les décisions concernant les données à préserver et ceux à éliminer sont basées sur une connaissance sémantique des données. Par exemple, une image peut être dégradée en éliminant des informations sur la couleur ou en réduisant la résolution. Une vidéo peut être dégradée en réduisant le nombre d'images par seconde. Dans tous les cas, la dégradation s'efforce de préserver les informations qui ont le plus de valeur sémantique.

- **La distillation spécifique aux données :** la distillation est une compression avec un facteur de perte très élevé qui préserve néanmoins la majeure partie du contenu. Elle peut être perçue comme une optimisation du contenu tout en respectant les contraintes de représentation. Le tableau 2.1 récapitule les axes de distillation relatifs à trois types de données : texte formaté, images et vidéo. La sémantique des données est logiquement indépendante de leur encodage. Par exemple, un document Postscript peut servir à encoder aussi bien un texte qu'une image. L'implémentation d'une méthode de distillation nécessite toutefois la connaissance parfaite de l'encodage.

- **Le raffinement :** le processus de raffinement permet de recevoir les données avec une qualité progressive, pouvant atteindre la qualité de la représentation d'origine. Dans un premier temps, l'utilisateur ne reçoit qu'un aperçu des données, lui permettant de se focaliser sur les aspects qui l'intéressent (par exemple, en zoomant sur une partie du graphique). Comme pour la distillation, le raffinement dépend de la sémantique des données et l'implémentation nécessite une connaissance parfaite de l'encodage.

Type sémantique	Certains types d'encodage	Axes de distillation
Image	GIF, JPEG, PPM, figure Postscript	résolution, couleur, profondeur, palette de couleur
Texte	plat, HTML, Postscript,PDF	formatage lourd, balisage
Vidéo	NV, H.261, VQ, MPEG	résolution, taux d'images, profondeur des couleurs, limite de progression

TAB. 2.1 – Axes de distillation associés à chaque type de données

Ci-après, nous présentons un projet qui met en œuvre la distillation à la demande.

Une plate-forme pour la distillation à la demande

Cette plate-forme de distillation [27] est constituée par les modules suivants : le mandataire, un ou plusieurs distillateurs spécifiques aux données, un moniteur réseau optionnel et une bibliothèque de support pour les applications (figure 2.2).

Le mandataire Le client communique exclusivement avec le mandataire. Dans un environnement réseau hétérogène, le mandataire doit se trouver près de la limite qui sépare la connectivité faible de la connectivité forte. Le mandataire recueille le contenu des serveurs Internet pour le compte du client. Il détermine les types de données de haut niveau correspondant à chaque composant et sélectionne le distillateur qui doit être employé. Quand le mandataire fait appel à un distillateur, il lui fournit certaines informations comme la configuration matérielle du client, les encodages acceptés et la bande passante réseau disponible.

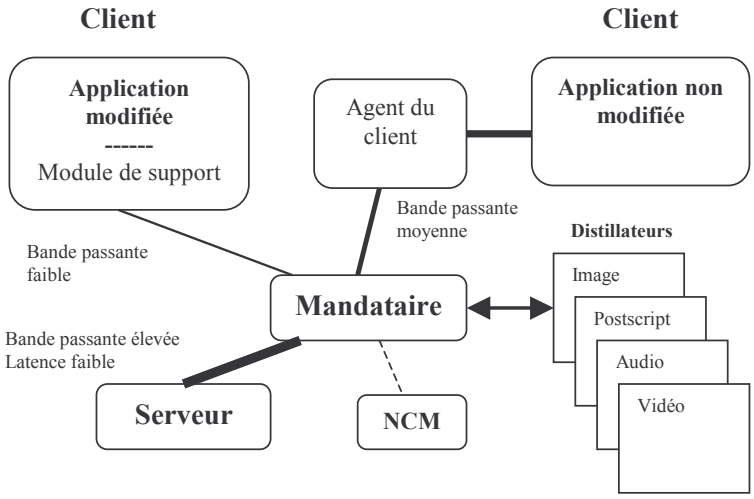


Fig. 2.2 – Architecture de la plate-forme de distillation

Les distillateurs Ce sont des processus contrôlés par le mandataire, effectuant la distillation et le raffinement pour le compte d'un ou de plusieurs clients. Les distillateurs utilisent les informations fournies par le client pour générer la meilleure représentation possible pour un client donné. Par exemple, si le client dispose d'un écran avec une résolution de 8 niveaux de gris, d'une bande passante de 9600 b/s et d'un temps d'acheminement de 4 s, le distillateur doit estimer le traitement nécessaire pour aboutir à une représentation de l'image de l'ordre de 37.5 kb.

Le moniteur réseau (NCM) Ce composant supervise les paramètres réseau, comme la bande passante réseau, et informe le mandataire de tout changement significatif de l'un de ces paramètres.

Le module de support du client Ce module fournit aux applications modifiées des méthodes pour manipuler les données et interagir avec le mandataire. Il se focalise sur la manière de recueillir les documents distillés, il spécifie les encodages que le client accepte et influence les décisions du mandataire en délimitant les contraintes sur les axes de distillation. En ce qui concerne les applications non modifiées, l'invocation du module de support se fait à travers un agent client. Cet agent fournit aux clients les données selon la forme qu'ils demandent. Du point de vue des applications, cet agent est équivalent à un mandataire distant, bien qu'il agisse en réalité comme un filtre permettant une communication efficace pour le compte des applications. Il existe certaines limites à cette approche car, par exemple, il est difficile de proposer une interface de raffinement pour la plupart des navigateurs Web existants.

2.3.3 Réplication des données

La réplication des données est l'une des techniques d'adaptation les plus utilisées dans les environnements mobiles, elle permet aux terminaux qui possèdent une faible connectivité de continuer d'opérer localement sur les données. Cependant, beaucoup de solutions de réplication existantes [18] ont été conçues pour des infrastructures fixes. Les connexions peuvent être temporaires mais l'adresse et l'identité des partenaires de la réplication demeurent inchangées. De plus, la synchronisation directe entre les hôtes mobiles n'est souvent pas permise ; cette solution est parfois la seule manière d'assurer la cohérence des données entre les hôtes dans un environnement diffus. Certains systèmes ont tenté de résoudre le problème en concevant des schémas de réplication de plusieurs à plusieurs. Généralement, trois facteurs caractérisent un bon modèle de réplication dans les environnements mobiles [83] :

- **La synchronisation de plusieurs à plusieurs :** il est généralement moins coûteux de communiquer avec les hôtes voisins. Il est également difficile de prévoir les hôtes qui seront localisés au même endroit à un moment donné. Les utilisateurs mobiles doivent donc avoir l'aptitude de synchroniser leurs données avec leurs voisins les plus proches, et même de propager les synchronisations d'un hôte vers l'autre. La cohérence peut, bien entendu, être maintenue, même si les hôtes ne peuvent se synchroniser entre eux, comme le démontre les modèles de réplication client/serveur. L'avantage est surtout en terme de coût et de performance. Dans un environnement diffus, c'est même souvent l'unique option.
- **Les facteurs de réplication :** la plupart des systèmes de réplication fournissent un nombre limité de réplicas d'un objet donné. En outre, ces systèmes ne passent pas très bien à l'échelle. Dans les environnements mobiles, les facteurs de réplication (nombre de réplicas) doivent être revus en hausse. La raison principale est que chaque utilisateur peut souhaiter garder des copies locales des données sur tous ses terminaux mobiles, ce qui a pour conséquence de décupler le facteur de réplication si l'utilisateur garde également des copies sur son ordinateur de bureau.
- **Les contrôles détaillés de la réplication :** par définition, un service de réplication fournit aux utilisateurs un certain degré de contrôle de la réplication - une méthode qui indique les objets qui doivent être répliqués. Beaucoup de systèmes répliquent les données avec une grande granularité, ce qui signifie, dans certains cas, qu'une partie importante des données sera inutilement téléchargée sur le terminal de l'utilisateur qui ne peut généralement pas stocker de larges volumes. La solution à adopter, dans ce cas, consiste à garantir le niveau de flexibilité approprié pour sélectionner individuellement les objets à répliquer.

Nous citons ci-après deux plates-formes de réplication destinées aux environnements mobiles : Roam et Bayou.

Roam

Roam [84] est un système de réplication, basé sur la notion de *Ward* (*Wide Area Replication Domain*). Il combine des éléments des modèles pair à pair (voir section 4.1.4) et client/serveur. Un *Ward* est une collection d'hôtes, localisés dans la même zone géographique et dotés d'une connexion faible. Bien que tous les membres d'un *Ward* soient des pairs égaux, ils désignent un

maître (*Ward Master*). Celui-ci est le seul lien qui relie un *Ward* avec les autres. Par conséquent, le *Ward Master* est la seule entité qui a connaissance des autres réplicas en dehors du *Ward*. Tous les *Ward Masters* appartiennent à un *Ward* de niveau supérieur, formant ainsi une hiérarchie à deux niveaux. Les *Ward Masters* agissent pour le compte du *Ward* en important et en exportant les mises à jour pour tout le *Ward*. La cohérence entre les réplicas est maintenue à travers la communication entre les *Ward Masters* et la propagation des mises à jour à l'intérieur du *Ward*. La figure 2.3 illustre l'architecture de base avec certains aspects que nous abordons dans la suite.

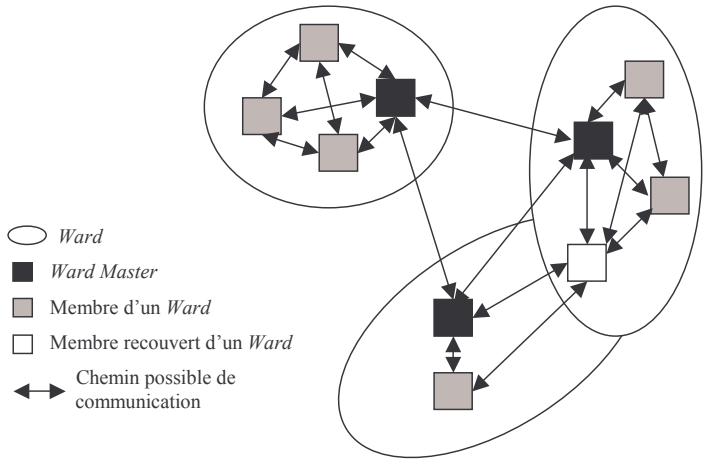


Fig. 2.3 – Architecture de base d'un Ward

La flexibilité du modèle La flexibilité de la réplication est l'une des caractéristiques importantes de ce modèle. L'espace des données stockées dans chaque *Ward* est ajustable dynamiquement. Quand les membres d'un *Ward* changent la composition des données qu'ils répliquent, la composition du *Ward* change aussi. De plus, chaque membre d'un *Ward*, y compris le *Ward Master*, peut stocker localement un sous-ensemble différent des données du *Ward*. Ce schéma de réplication, appelé la réplication sélective, améliore l'efficacité et l'utilisation des ressources [83].

Prise en charge de la mobilité La mobilité se produit quand un hôte change de zone géographique et rentre en contact avec des hôtes appartenant à un autre *Ward*. Cette mobilité entraîne un nouveau problème de réplication car les deux *Wards* n'ont pas forcément des espaces de données identiques. Roam définit deux types de mobilité : la mobilité à long terme et la mobilité à court terme. Ces deux types de mobilité sont pris en charge respectivement par deux opérations : le changement du *Ward* et le recouvrement du *Ward*.

1. **Le changement de Ward :** il implique un changement à long terme, peut-être permanent, dans la composition du *Ward*. L'hôte mobile doit changer de « *Home Ward* » (*Ward* de base), alors que les autres participants à l'ancien et au nouveau *Wards* doivent mettre à jour la liste des participants. Les changements dans l'espace du *Ward* sont propagés vers les autres *Wards* d'une manière optimiste, de façon à ce que seuls les *Ward Masters* intéressés les prennent en compte.

2. **Le recouvrement du Ward :** contrairement au changement, qui est une opération lourde, le recouvrement est une opération légère qui n'entraîne pas des modifications globales à l'intérieur du système. Le nouveau *Ward* est le seul concerné par cette opération. Le recouvrement permet à un hôte d'appartenir simultanément à plusieurs *Wards*. L'espace de donnée de ce nouveau *Ward* n'est pas altéré, car le nouveau membre se fait attribuer un statut particulier. La seule différence avec un membre à part entière est dans la gestion de l'espace de données. Au lieu de fusionner cet espace avec les données stockées sur l'hôte mobile, il demeure inchangé. Les données partagées entre l'hôte mobile et l'espace de données peuvent être réconciliées localement au niveau de chaque membre du *Ward*. Cependant, les données qui ne font pas partie de l'espace de données ne peuvent pas être réconciliées. Elles doivent rester temporairement non synchronisées ou, sinon, être synchronisées avec le « *Home Ward* ».

Bayou

Bayou [100] est un système qui gère les conflits produits par les activités concurrentes de plusieurs entités. Il tolère que les données soient modifiées de manière concurrente en maintenant une faible cohérence entre les différents réplicas. Les opérations de lecture et d'écriture sont autorisées sans recours à une coordination explicite avec les autres réplicas. Chacun d'eux reçoit les mises à jour, effectuée par ailleurs, soit directement, soit indirectement à travers une chaîne de propagation des mises à jour.

La particularité de Bayou est que les applications peuvent être impliquées dans la détection et la résolution des conflits, si les données dépendent de la sémantique de l'application. Les applications ont la possibilité de spécifier au système leur propre notion du conflit et de fournir une politique pour le résoudre. Un mécanisme de détection automatique, appelé « vérification de dépendance », est mis en œuvre à chaque fois qu'une opération d'écriture est propagée au serveur. Il consiste à tenter de reproduire la même opération au niveau du serveur et de vérifier si le résultat correspond à celui qui est attendu. Si ce n'est pas le cas, le serveur invoque une procédure pour résoudre le conflit. Cette procédure est fournie par les développeurs de l'application. Bayou nécessite la présence d'un serveur pour gérer les opérations de mise à jour, ce qui convient plutôt à un environnement nomade qu'à un environnement diffus.

2.3.4 Gestion de l'énergie

L'énergie est une ressource critique sur les terminaux mobiles. Malgré des progrès indéniables dans la conception de circuits peu gourmands en énergie et de batteries dont la durée de vie est de plus en plus longue, l'autonomie des terminaux mobiles reste limitée. Les travaux de recherche récents ont démontré que la gestion de l'énergie doit concerner également des niveaux plus élevés que le système d'exploitation tout seul [24]. Par exemple, une forte compression des données peut économiser une quantité considérable de l'énergie, si la transmission sur le réseau est coûteuse de ce point de vue.

Les progrès dans la conception de programmes sensibles à l'énergie dépendent de la possibilité d'attribuer la consommation de l'énergie à des composants logiciels particuliers, de la même

manière que les outils *prof* et *gprof* aident à identifier les composants qui sont gourmands en consommation de CPU. PowerScope [25] est un outil qui associe la consommation de l'énergie à la structure d'un programme. Il permet de déterminer les fractions de l'énergie totale, consommée pendant une certaine durée, qui sont dues à certains processus dans le système. L'unité mesurée peut être affinée si on souhaite mesurer la consommation de certaines procédures dans le système.

Lorch et al. [60] classifient les techniques logicielles pour économiser l'énergie en trois catégories : transition, modification de la charge et adaptation. La stratégie de transition concerne le moment où il faut basculer vers un mode qui se caractérise par une faible consommation de l'énergie et des fonctionnalités réduites. Cette stratégie nécessite deux types d'informations à propos d'un composant : la connaissance des caractéristiques de chaque mode de fonctionnement et la connaissance de ces besoins pour son fonctionnement futur. Les caractéristiques du mode désignent les avantages et les inconvénients de chaque mode (l'énergie économisée, les fonctionnalités sacrifiées, le temps nécessaire pour basculer vers ce mode, etc.). La stratégie de modification de la charge concerne la modification du niveau d'utilisation d'un composant afin qu'il puisse basculer vers un mode d'économie d'énergie. La stratégie d'adaptation concerne le changement du composant par un autre qui consomme moins d'énergie.

Un exemple de la mise en œuvre de ces stratégies est le stockage secondaire dans les ordinateurs portables. Il consiste généralement en un disque magnétique et une mémoire SDRAM qui est utilisée comme un cache du disque. Ce cache améliore la performance globale du système et réduit la consommation de l'énergie en réduisant les accès au disque dur qui consomme beaucoup plus d'énergie que la mémoire SDRAM.

L'une des stratégies de transition possibles pour les disques durs est de décider le basculement au mode « sleep » après un certain délai d'inactivité. Ce mode permet d'économiser l'énergie en arrêtant la rotation du disque dur et en réduisant au minimum l'activité du contrôleur de disque. Certains systèmes fixent ce délai de manière dynamique suivant la fréquence d'accès au disque, car des arrêts trop fréquents peuvent augmenter la consommation de l'énergie, compte tenu du coût nécessaire à la remise en marche. Un autre type de stratégie est de modifier la charge de travail du disque dur en modifiant sa configuration. L'augmentation de la taille du cache peut conduire à une réduction de la consommation d'énergie. Le préchargement est également une stratégie qui peut être utilisée dans ce but. Si le cache est occupé, avant l'arrêt des disques, par les données qui vont être vraisemblablement utilisées dans le futur, la nécessite de remettre le disque en marche s'en trouve un peu plus retardée.

Enfin, en ce qui concerne les stratégies d'adaptation possibles, l'utilisation d'une mémoire Flash à la place du disque dur a l'avantage de réduire sensiblement la consommation de l'énergie et même de réduire les temps d'accès en lecture. C'est généralement le cas pour les terminaux légers comme les PDA. L'inconvénient de cette stratégie est la capacité des mémoires Flash qui reste bien inférieure à celle des disques durs.

2.4 Synthèse

Ce chapitre avait pour but d'étudier les problématiques des environnements nomades et diffus. Cette étude montre le caractère très particulier de ces environnements et aussi la diversité

des solutions proposées. Ces solutions vont d'une réadaptation d'idées, qui existaient déjà dans d'autres types d'environnement, à des idées totalement nouvelles.

La liste des problématiques n'est bien entendu pas exhaustive, nous nous sommes penchés essentiellement sur les aspects qui relèvent du cadre de cette thèse. Par ailleurs, nous avons tenté de fournir une vision qui soit la plus indépendante possible par rapport à un niveau particulier des systèmes mobiles. D'autres problématiques et techniques sont traitées dans la partie consacrée aux intergiciels mobiles. C'est notamment le cas en ce qui concerne les techniques d'adaptation.

L'étude menée sur l'adaptabilité montre l'omniprésence de ce concept dans différents aspects des systèmes nomades et diffus. Il se révèle donc indispensable pour une gestion efficace des différentes problématiques citées. Nous avons mis l'accent sur les techniques d'adaptation concernant la gestion du flux de données et de l'énergie. Ces techniques font l'un des objets de notre contribution.

Chapitre 3

Les intergiciels fixes

Les intergiciels sont des composants logiciels qui visent à mettre en œuvre un modèle de répartition entre plusieurs nœuds interconnectés à travers un réseau de communication. L'utilisation d'un intergiciel par rapport à l'utilisation directe des primitives de communication fournies par le système d'exploitation, permet l'intégration d'un modèle de répartition sophistiqué et offre des abstractions de haut niveau. Ce modèle est un ensemble de sémantiques qui décrivent les interactions entre les entités d'un point de vue logique. Les abstractions fournissent des solutions à des problèmes fréquemment rencontrés lors du développement d'applications réparties, comme les problèmes relatifs au transport de messages, à la représentation des données, à la coordination entre les nœuds ou à la gestion de transactions. Les intergiciels permettent ainsi aux développeurs de se décharger de ces difficultés et de se consacrer essentiellement aux fonctionnalités propres de leurs applications.

Ce chapitre fournit en premier lieu un aperçu des différents types d'intergiciels conçus pour les environnements fixes. Nous examinons en particulier le cas des intergiciels orientés message. Nous décrivons ensuite la spécification *Java Message Service* qui concerne la conception d'intergiciels orientés message interopérables. Enfin, nous présentons la plate-forme générique et extensible Jonathan. Cette plate-forme fournit des briques de base pour la conception d'intergiciels qui adoptent différents modèles de répartition et peuvent être adaptés à différents contextes. Elle est exploitée par notre intergiciel.

3.1 Généralités

D'un point de vue architectural, un intergiciel est une couche logicielle qui se situe entre le système d'exploitation et les applications. L'intergiciel masque les détails d'implémentation, ayant trait à l'hétérogénéité des plates-formes d'exécution, à travers un ensemble d'interfaces. Ces interfaces sont construites à partir les services de base du système d'exploitation et de la plate-forme matérielle sous-jacente.

Les intergiciels se distinguent par une grande diversité en ce qui concerne leur architecture et les services qu'ils proposent. Nous résumons ci-dessous quelques-uns des services retrouvés

fréquemment.

- **Communication :** les différents composants d'un système réparti peuvent résider sur des nœuds différents et communiquent par conséquent à travers le réseau. L'intergiciel fournit un service pour le transport de messages d'un nœud vers l'autre. Ces services sont construits en utilisant les primitives de transport du système d'exploitation, comme les sockets TCP ou UDP.
- **Adressage :** les composants répartis doivent pouvoir s'identifier pour pouvoir interagir. L'intergiciel attribue à chaque composant un identifiant qui peut être échangé avec les autres nœuds. Cet identifiant désigne en général le nœud qui héberge le composant ainsi qu'un nom qui permet de distinguer le composant au sein du nœud.
- **Représentation des données :** les composants répartis peuvent s'échanger des données ayant des structures complexes. Ces structures doivent être codées selon un format qui soit transmissible par un protocole réseau (une séquence d'octets). Cette transformation est connue sous le nom d'emballage de données (*marshalling*), alors que la transformation inverse est connue sous le nom de déballage de données (*unmarshalling*). Ces transformations sont susceptibles de générer beaucoup d'erreurs lors du développement, et sont donc fournies habituellement par l'intergiciel.
- **Coordination et exécution :** le parallélisme qui existe entre les activités des composants répartis nécessite une synchronisation lorsqu'ils communiquent entre eux. Cette communication doit être synchronisée avec l'exécution des différentes tâches par les composants. La synchronisation peut se faire de différentes manières. Un composant peut interrompre son exécution jusqu'à ce que le composant distant termine l'exécution du service demandé (forme synchrone de coordination). Un composant peut envoyer une requête à un autre composant et continuer son exécution en attendant la réponse (forme asynchrone de coordination).
- **Liaison :** l'objectif de la liaison est d'interconnecter un ensemble d'objets répartis en définissant un chemin d'accès à travers lequel un objet peut être accédé par un autre objet. Les informations permettant d'établir cette liaison sont regroupées dans une référence que fournit l'intergiciel. Les ressources nécessaires à cette liaison sont également allouées par l'intergiciel.
- **Fiabilité :** les protocoles réseau possèdent différents niveaux de fiabilité. Certains protocoles ne garantissent pas que tous les paquets, envoyés par un composant, soient reçus dans le bon ordre par le destinataire. Certains intergiciels comportent des mécanismes pour la détection et la correction d'erreurs ou pour la retransmission des messages. Ces mécanismes servent à assurer un certain niveau de fiabilité. Parmi les niveaux définis dans le domaine des systèmes répartis : le meilleur effort (*best effort*), au plus une fois (*at-most-once*), au moins une fois (*at-least-once*), et une et une seule fois (*exacly-once*).
- **Hétérogénéité :** les intergiciels ont pour objectif de masquer l'hétérogénéité qui existent entre les plates-formes matérielles, les systèmes d'exploitation, les protocoles réseau et les langages de programmation, en offrant une interface commune au développeur. Cette hétérogénéité peut être le résultat de la diversité des intergiciels eux même ou des composants applicatifs qu'ils prennent en charge. Les intergiciels génériques [82] proposent un modèle qui permet de faire interagir des composants écrits pour des intergiciels différents.

3.2 Types d'intergiciels

Nous présentons, dans cette section, les principaux types d'intergiciels existants. Cette classification est basée sur le modèle de répartition et l'architecture de chaque type. Nous traitons avec plus de détails les intergiciels orientés message dans la section 3.3.

3.2.1 Les intergiciels procéduraux

Les intergiciels procéduraux adoptent le modèle d'appel de procédure à distance (RPC) [7], ils fournissent un environnement structuré pour qu'un client puisse appeler une procédure qui peut s'exécuter sur un autre nœud du réseau, appelé serveur. L'intergiciel du côté client implémente l'appel de procédure en emballant les paramètres dans un message qui est envoyé au serveur. L'intergiciel du côté serveur extrait ces paramètres après avoir déballé le message et exécute la procédure appelée. Les résultats sont ensuite emballés dans un message et transmis au client.

Les opérations d'emballage et de déballage des données sont implémentées par une souche au niveau du client et un squelette au niveau du serveur. Ces modules sont générés automatiquement par le compilateur RPC à partir de la signature de la procédure à appeler. La souche présente une interface identique à celle de la procédure distante, mais remplace le traitement par la construction et l'envoi d'un message. Le squelette, qui attend en permanence l'arrivée de messages demandant l'exécution d'une procédure, déballe les paramètres, fait appel au corps de la procédure à exécuter, emballe et envoie les résultats au client.

Les intergiciels procéduraux adoptent un mode d'interaction synchrone. Les procédures sont exécutées avec la sémantique « au plus une fois ». Si l'exécution échoue, l'intergiciel renvoie une exception.

3.2.2 Les intergiciels orientés objets

Les intergiciels orientés objets sont une évolution des intergiciels procéduraux. L'idée principale est de transposer les principes de la programmation orientée objet dans le domaine réparti. Les différents objets d'une application sont répartis à travers plusieurs nœuds. Tout client, qui possède une référence sur un objet serveur, peut envoyer une requête pour l'exécution d'une méthode de cet objet. Les opérations d'emballage et de déballage sont effectuées par les souches et les squelettes générés à partir de la définition de l'interface de l'objet appelé.

Les références d'objets sont des informations que se partagent tous les nœuds de l'application, permettant de désigner sans ambiguïté chaque objet. La référence associe l'adresse réseau du nœud hébergeant l'objet avec l'identifiant de l'objet sur ce nœud. Les intergiciels orientés objets prévoient généralement un service de nommage qui permet de diffuser les références d'objets. Ce service, sous forme d'un répertoire global, associe le nom de l'objet avec sa référence. Il évite donc aux objets de se référencer explicitement. Le déplacement d'un objet serveur d'un nœud vers un autre n'influe pas sur le corps des objets clients. Seule la référence de l'objet serveur est modifiée.

Parmi les intergiciels orientés objets, nous pouvons citer CORBA de l'OMG [79], le modèle COM/DCOM de Microsoft [64] et Java RMI [94]. Le mode d'interaction par défaut de ces intergiciels reste le mode synchrone. L'objet client est bloqué jusqu'à ce que l'invocation de la méthode sur le serveur soit terminée. Cependant, d'autres modes d'interaction sont également pris en charge. Le standard CORBA définit, par exemple, le mode d'interaction asynchrone ou synchrone différé où le client peut reprendre l'exécution avant même que la réponse du serveur ne soit arrivée.

3.2.3 Les intergiciels transactionnels

Les intergiciels transactionnels, appelés également moniteurs transactionnels, prennent en charge des transactions exécutées par des composants situés sur plusieurs nœuds. Ces intergiciels utilisent le protocole de validation en deux phases pour implémenter les transactions réparties. Parmi les intergiciels transactionnels, citons Tuxedo de BEA [6] et CICS d'IBM [43].

Les composants peuvent utiliser les modes d'interaction synchrone ou asynchrone. Un composant peut regrouper plus d'une requête dans une transaction. Les transactions peuvent aussi être assurées en utilisant les services de systèmes de gestion de bases de données, ce qui améliore sensiblement la tolérance aux fautes. De plus, la plupart des intergiciels transactionnels fournissent des services de réplication et d'équilibrage de charge.

Cependant, certains intergiciels transactionnels n'incluent pas de services d'emballage de données. Cette tâche peut se révéler fastidieuse et délicate pour les développeurs d'applications. En outre, bien que le protocole de transaction (validation en deux phases) soit standardisé, il n'existe pas à l'heure actuelle une approche standard pour la définition des services.

3.3 Les intergiciels orientés messages (MOMs)

Les intergiciels orientés messages (MOMs) trouvent leurs origines dans les modèles publication/souscription et les systèmes de notifications d'évènements [4]. Les premières implémentations sont apparues au milieu des années 1980. Ces intergiciels peuvent également être désignés par les intergiciels basés sur les évènements. En effet, les concepts de message et d'événement sont assez proches, car un message peut être considéré comme une notification d'évènement.

Le mode de communication des MOMs est basé sur l'échange asynchrone et fiable de messages. Les MOMs substituent, au modèle d'interaction synchrone classique (client/serveur), un modèle proche de celui du pair à pair, où chaque entité communicante peut envoyer et recevoir des messages des autres entités. Ce mode de communication par messages améliore la flexibilité des applications car les entités ne sont plus couplées les unes aux autres. L'émetteur d'un message peut poursuivre son exécution dès l'envoi du message.

Outre la flexibilité, les MOMs ont démontré leur capacité à répondre aux besoins de passage à l'échelle des applications réparties très complexes. La modification ou l'ajout de nouvelles entités à l'application peut se faire de manière relativement aisée car il n'existe pas de dépendances

fortes entre les composants. L'échange de messages peut également s'effectuer entre deux entités sans connaître forcément l'adresse ou le type de plate-forme sur laquelle s'exécute le destinataire des messages.

Les MOMs fournissent une prise en charge appropriée des applications basées sur les évènements. Quand un événement se produit, le client délègue au MOM la responsabilité d'informer le serveur qu'une action doit être déclenchée. Actuellement, les MOMs sont généralement utilisés pour l'intégration de différents types d'applications et de données dans les systèmes d'information d'entreprise. Le faible couplage entre les entités communicantes dans un MOM, en terme de disponibilité et de logique fonctionnelle, permet d'incorporer à un système d'information global les applications et les données patrimoniales de l'entreprise. Il ouvre la voie à de nouveaux types d'accès comme le Web. L'échange des messages peut se faire dans un contexte interne (pour le EAI) ou externe (pour le B2B ou le B2C) à l'entreprise. Les MOMs sont également dotés de services sophistiqués, comme le routage de messages ou la transformation du contenu, qui leur permettent de s'adapter à une population hétérogène et variable d'applications et d'utilisateurs.

Parmi les MOMs industriels, citons : MQSeries d'IBM intégré dans la plate-forme WebSphere [44], MSMQ de Microsoft [65] ou TIB/Rendezvous de Tibco [101]. Il existe également des MOMs dans le domaine du logiciel libre : JORAM [72] ou OMSQ [68].

Le mode de communication

Les MOMs définissent plusieurs modes d'échange de messages entre les entités :

- **Echange simple :** les messages sont envoyés directement à leur destinataire. Ce dernier doit être disponible au moment de la réception afin de pouvoir les recevoir. Ce mode de communication s'apparente de celui des systèmes pair à pair.
- **Files de messages :** les messages ne sont envoyés qu'à un seul destinataire à la fois. Ils sont d'abord envoyés et stockés dans une file qui tient le rôle d'une boite aux lettres du destinataire. Cette file de messages peut faire partie du même processus que celui du destinataire, elle peut également constituer une entité intermédiaire qui se situe sur un autre nœud. Cette dernière solution présente l'avantage de fournir plus de fiabilité. La file permet de stocker les messages temporairement dans le cas où leur destinataire ne serait pas disponible (arrêt de la machine, destinataire surchargé, etc.). Sur certains MOMs, les messages contenus dans la file peuvent être stockés de manière persistante (sur un support stable) permettant ainsi de prévenir leur perte en cas d'arrêt de la machine. Ce cas de figure permet d'assurer un niveau de fiabilité de type « une et une seule fois ».
- **Publication/souscription (*publish/subscribe*) :** les messages sont envoyés à une entité intermédiaire, chargée de les remettre aux destinataires qui se sont abonnés à ce type de message. Ce mode présente l'avantage de pouvoir diffuser un message à plusieurs destinataires à la fois. La souscription peut se faire selon deux critères :
  - *Le sujet* : chaque message est classé selon son appartenance à un sujet prédéfini. Chaque entité doit explicitement s'abonner auprès du nœud intermédiaire (le fournisseur), pour pouvoir recevoir les messages liés à ce sujet.

– *Le contenu* : la souscription basée sur le contenu offre plus de souplesse, car la réception des messages peut se faire suivant des attributs liés à leur contenu même. Ce type de souscription permet également d'atteindre des destinataires suivant des critères plus précis.

3.4 La spécification JMS

JMS, acronyme de Java Message Service [36], est un ensemble d'interfaces applicatives (API) qui permettent le développement d'applications basées sur l'échange de messages. Ces APIs représentent les fonctionnalités communes des systèmes de messagerie actuels. Quand la spécification JMS fut conçue en 1998, son objectif principal était de résoudre les problèmes d'incompatibilité entre les différents produits MOM existants, en fournissant une API commune. Depuis ce moment, plusieurs éditeurs de logiciels ont fourni une implémentation complète de cette API. Le tableau 3.1 résume quelques-unes des plates-formes industrielles qui adoptent la spécification JMS. En outre, JMS a été intégré à la plate-forme J2EE (*Java 2 Entreprise Edition*) à partir de la version 1.3 mais il reste utilisable avec les versions antérieures en utilisant un outil externe qui implémente l'API.

Les systèmes de messagerie sont basés sur une conception pair à pair, tous les utilisateurs de JMS sont donc désignés comme des clients. Une application JMS est formée d'un ensemble de messages définis par l'application et un ensemble de clients qui s'échangent ces messages. L'architecture d'un système JMS est organisée autour de clients et d'un fournisseur (*Provider*). Le fournisseur est le nœud de transition de tous les messages. Il assure des fonctions comme le routage des messages, la gestion des priorités ou la persistance.

En se basant sur le passage des messages, JMS met en oeuvre un style de communication faiblement couplé et asynchrone. Le fournisseur JMS transmet les messages aux clients suivant deux modes :

– Synchrone : le client interroge le fournisseur s'il possède un message qui lui est destiné en désignant un emplacement donné (voir section 3.4.2).

Produit	**Editeur**	**Lien Web**
BEA WebLogic Platform	BEA	www.bea.com
Websphere MQ	IBM	www-306.ibm.com/software/integration/mqfamily/api/mqjava.html
Sonic MQ	Sonic Software	www.sonicsoftware.com/products/sonicmq/technical_overview/index.ssp
SpiritWave	SpiritSoft	www.spirit-soft.com/products/wave/introducing.shtml
Java Message Queue	Sun	wwws.sun.com/software/products/message_queue/index.html
Entreprise Message Service	Tibco	www.tibco.com/software/enterprise_backbone/enterprise_jms.jsp

Tab. 3.1 – Quelques produits compatibles JMS

– Asynchrone : le fournisseur envoie les messages aux clients dès leur réception.

Par ailleurs, JMS permet de garantir qu'un message n'est délivré qu'une et une seule fois. Il fournit des mécanismes qui permettent de s'assurer que tous les messages arrivent à destination et empêchent la réception de messages dupliqués.

3.4.1 L'architecture d'un système JMS

Une application JMS est composée des parties suivantes (figure 3.1) :

– *Le fournisseur JMS* est le système de messagerie qui implémente les interfaces JMS fournissant les mécanismes pour le contrôle et l'administration du système.
– *Les clients JMS* sont les composants, écrits dans le langage Java, qui produisent et consomment les messages.
– *Les Messages* sont les éléments qui communiquent l'information entre les clients JMS.
– *Les objets administrés* sont des objets JMS configurés à l'avance et créés par un administrateur pour l'usage des clients. Il existe deux types d'objets administrés : les destinations et les usines d'objets (voir section 3.4.3).
– *Les clients natifs* sont des composants qui utilisent une API MOM autre que celle de JMS. JMS fournit des passerelles permettant de traduire les requêtes de ces composants en requêtes JMS.

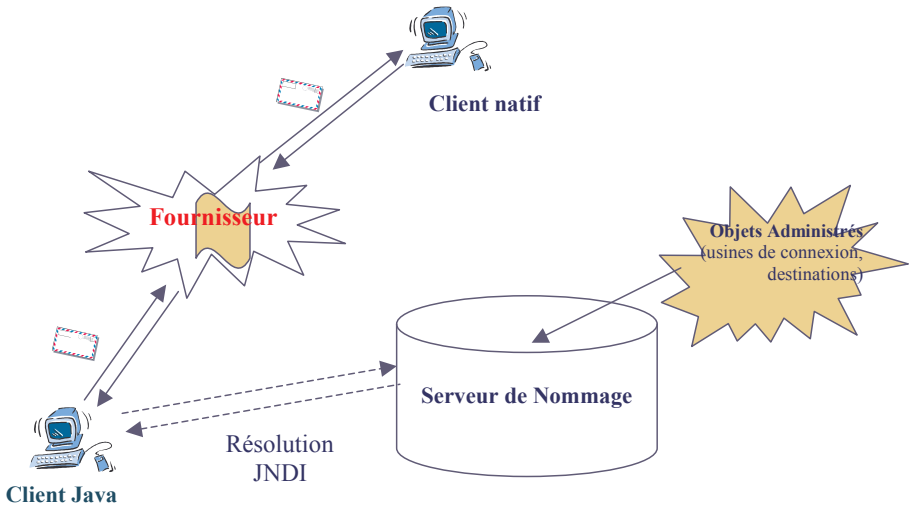


Fig. 3.1 – Architecture d'un système JMS

3.4.2 Les domaines de messagerie

La spécification JMS propose deux domaines de messageries séparés

Le domaine point à point (PTP)

Ce domaine est basé sur le concept de files de messages, d'émetteurs et de récepteurs de messages. Chaque message est adressé à une file d'attente (*queue*) spécifique au niveau du fournisseur. Le récepteur retrouve le message au niveau de cette file et envoie un acquittement au fournisseur si le traitement du message s'est déroulé correctement.

Fig. 3.2 – Le mode point à point de JMS

Les files d'attente gardent tous les messages qui leurs sont adressés jusqu'à ce qu'ils soient consommés ou que leur délai de validité expire. Ces messages peuvent être reclassés par le fournisseur selon leur priorité. Leur ordre de réception n'est donc pas toujours identique à leur ordre d'envoi. Notons que chaque message ne peut avoir qu'un et un seul récepteur.

Par ailleurs, l'émetteur et le récepteur d'un message ne possèdent pas de dépendances temporelles. Le récepteur doit recevoir son message même dans le cas où il ne serait pas disponible au moment de l'émission du message. La figure 3.2 illustre l'envoi d'un message vers une file. Le récepteur doit acquitter la réception du message auprès du fournisseur.

Le domaine publication/souscription (*Publish/Subscribe*)

Ce domaine repose sur le concept de sujets (*topics*). Un message est adressé à un sujet particulier au niveau du fournisseur et peut avoir plusieurs consommateurs qui sont anonymes pour le producteur du message.

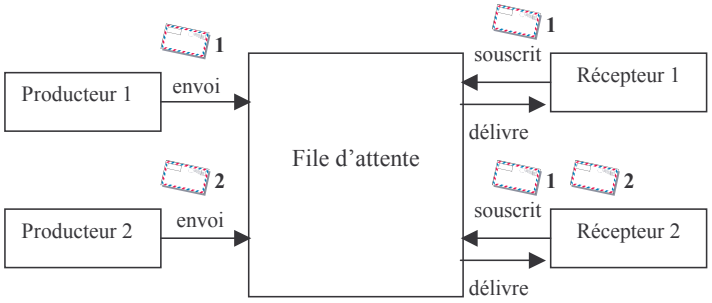


Fig. 3.3 – Le domaine publication/souscription de JMS

Un consommateur ne peut recevoir de messages, concernant un sujet donné, qu'après s'être

préalablement souscrit. Il existe par conséquent une dépendance temporelle. Les consommateurs ne reçoivent pas les messages publiés avant leur souscription et doivent rester actifs pour recevoir tous les messages. JMS assouplit néanmoins cette dépendance temporelle à travers le mécanisme des souscriptions durables. Celles-ci permettent la réception de messages envoyés même à un moment où les consommateurs n'étaient pas actifs. Ils fournissent la flexibilité et la fiabilité du mode point à point tout en permettant l'envoi de messages à plusieurs clients. La figure 3.3 illustre l'envoi par deux producteurs de messagers vers un sujet. Ces messages sont délivrés à deux consommateurs après leur souscription.

3.4.3 Quelques objets de l'API JMS

- **Usine de connexion :** le client utilise cet objet pour créer une connexion avec un fournisseur. Elle encapsule un ensemble de paramètres de configuration.
- **Destination** : le client utilise cet objet pour spécifier la destination des messages qu'il produit (file d'attente ou sujet).
- **Connexion :** elle encapsule une connexion virtuelle avec le fournisseur JMS. Elle peut représenter, par exemple, une socket TCP/IP ouverte.
- **Session :** elle représente un contexte ordonné pour la production et la consommation des messages. Les sessions sont utilisées pour créer les producteurs de messages, les consommateurs de message et les messages eux mêmes. Elles fournissent également un contexte transactionnel pour le regroupement d'un ensemble d'envoi et de réception de messages dans une entité atomique.
- **Producteur de messages :** il est créé par une session et utilisé pour envoyer des messages à une destination.
- **Consommateur de messages :** il est créé par une session et utilisé pour recevoir des messages à partir d'une destination.
- **Sélection de messages :** la sélection est utilisée pour filtrer les messages reçus suivant le contenu de l'en-tête et des propriétés du message. Cette chaîne de caractères contient une expression SQL92 qui permet de filtrer la réception des messages selon des expressions conditionnelles.
- **Message :** il est constitué d'un entête, de propriétés et d'un corps.
  - *En-tête* : il contient un nombre de champs prédéfinis qui permettent d'identifier et de router le message (destination, mode de livraison, délai d'expiration, priorité, etc.)
  - *Propriétés* : elles permettent la compatibilité avec les autres MOMs et la création du sélecteur de messages.
  - *Corps* : il contient les données. Cinq formats sont définis : texte, objet, paires de nom et valeur, octets, flux.

3.4.4 Mécanismes de fiabilité

JMS définit quelques mécanismes pour garantir la transmission de messages et contrôler leur acheminement.

1. **Contrôle de l'acquittement des messages :** plusieurs niveaux de contrôle sont définis sur l'acquittement des messages.
   - Session.AUTO_ACKNOWLEDGE : l'acquittement s'effectue automatiquement par la session lorsque le message est reçu.
   - Session.CLIENT_ACKNOWLEDGE : le client acquitte explicitement le message en appelant sa méthode *acknowledge*.
   - Session.DUP_OK_ACKNOWLEDGE : cette option indique à la session que l'acquittement des messages peut se faire de manière plus laxiste. Ceci peut entraîner la réception de messages dupliqués, si le fournisseur JMS subit une défaillance par exemple.

2. **Persistance du message :** deux modes sont définis pour la persistance des messages :
   - Persistant (par défaut) : le fournisseur JMS doit s'assurer que le message n'est pas perdu en cas d'une défaillance. Le message est stocké sur un support stable.
   - Non persistant : le message n'est pas nécessairement stocké sur un support stable.

3. **Priorité des messages :** des niveaux de priorités de 0 à 9 sont définis pour chaque message. 9 est la priorité la plus élevée et 4 est la priorité par défaut.

4. **Délai d'expiration du message :** par défaut, un message qui n'a pas encore été envoyé à son destinataire, reste valide pendant un temps indéfini au niveau du fournisseur. Si un délai d'expiration est défini, le message peut être supprimé même avant son envoi.

5. **Destinations temporaires :** les destinations sont généralement créées de manière administrative plutôt qu'à travers un programme. JMS permet de créer des destinations dont la durée de vie ne dépasse pas celle de la connexion JMS qui les a créées. Les seuls consommateurs, qui peuvent recevoir les messages d'une destination temporaire, sont ceux créés par cette même connexion. Ce mécanisme permet, par exemple, à un client JMS de stocker provisoirement des messages au niveau du fournisseur.

3.4.5 Scénarios d'utilisation de JMS

Cette section présente deux scénarios d'utilisation d'un MOM JMS.

Scénario 1

Considérons le cas où le centre de données d'une banque doit fournir des informations concernant les nouveaux taux d'intérêts qui doivent être pratiqués. L'action immédiate à effectuer consiste à diffuser ces informations aux applications utilisées par les différentes agences. Avec JMS, ces informations peuvent être diffusées sous forme de messages en les envoyant à un sujet au niveau du fournisseur JMS. Ces messages sont reçus par les applications clients à travers une souscription à ce sujet (*topic*).

L'avantage immédiat de l'utilisation de JMS dans cette application est de garantir la réception de ces informations par tous les clients concernés. Et cela même dans le cas où un le lien de communication, entre les agences et le centre de données, serait momentanément défaillant. Les avantages à long terme sont plus considérables. Nous pouvons imaginer, qu'après quelques mois,

un service de la banque manifeste son intérêt pour les informations sur les taux afin d'établir un bilan périodique pour la direction. Par ailleurs, un partenaire de la banque peut indiquer des nouveaux taux à travers l'Extranet de la banque. Chacune de ces applications va effectuer des traitements différents sur les informations publiées, mais les obtiennent simplement en s'abonnant au sujet. Nous pouvons noter que le centre de données ne doit subir aucune modification pour prendre en charge ces nouveaux usages. Il n'est même pas obligé d'être informé de l'existence de ces nouvelles applications. La séparation, entre la transmission d'un événement et les sémantiques de son traitement, permet aux messages d'être utilisés de manières qui n'étaient pas prévues au départ.

Scénario 2

Considérons le système d'enregistrement en ligne d'un portail Internet. Ce système doit traiter des requêtes de création d'un nouvel utilisateur, ou de l'accès et de l'authentification d'un utilisateur existant. Un système de ce type doit gérer les requêtes qui peuvent être adressées par des milliers d'utilisateurs. L'utilisation d'une seule base de données centrale ne permet pas de prendre en charge tout le trafic. Les données concernant les utilisateurs doivent donc être répliquées sur plusieurs bases de données.

L'utilisation de JMS permet dans ce cas de répartir les requêtes des utilisateurs sur les différentes entités qui gèrent l'accès aux bases de données. Les requêtes sont envoyées à un sujet du fournisseur. Chaque entité récupère une requête de ce sujet dès qu'elle termine le traitement de la requête précédente. Cette architecture permet donc un meilleur passage à l'échelle grâce notamment à la répartition dynamique des requêtes.

3.5 Etude d'une plate-forme générique : Jonathan

Jonathan [23][73], développé à l'origine à France Telecom, est une plate-forme minimale et modulaire pour le développement des ORBs qui peuvent être considérés comme les composants centraux des intergiciels orientés objets. Jonathan est conçu avec l'idée d'introduire plus d'ouverture et de flexibilité aux intergiciels traditionnels qui se caractérisent par une certaine rigidité [82]. Il offre des interfaces et des composants qui peuvent être assemblés et enrichis afin de construire des ORBs, adaptés à des contraintes d'exécution spécifiques et dotés de politiques spécifiques de gestion de ressources. Sa structure interne est exposée dans le but de gérer ces problèmes en modifiant un minimum de code.

Jonathan fournit des composants réutilisables qui réalisent des tâches comme la communication entre les objets, la gestion de la mémoire ou l'emballage des données. Ces composants forment une couche abstraite qui sert de base à l'implémentation de personnalités spécifiques. Une personnalité est un ensemble d'interfaces applicatives normalisées (API) représentant une technologie intergicielle spécifique. Jonathan fournit à l'origine deux personnalités :

- David : une implémentation de l'ORB CORBA.
- Jeremie : une implémentation du modèle Java/RMI.

Jonathan est organisé autour de quatre composants implémentés en Java qui gèrent chacun des fonctions spécifiques.

- **Le composant de liaison :** il fournit des outils pour la gestion des noms (identificateurs) et le développement des usines de liaisons ou l'extension de celles qui existent déjà. Ce composant permet l'introduction de mécanismes de liaisons arbitraires entre les objets. Parmi les formes de liaisons possibles, nous pouvons citer les liaisons multimédia pour le transport de flux audio ou vidéo avec différentes qualités de service, ou les liaisons pour les communications de groupe.
- **Le composant de communication :** il définit les interfaces des composants impliqués dans la communication inter-objets comme les protocoles et les sessions.
- **Le composant de ressources :** il définit les abstractions pour la gestion des ressources utilisées par l'ORB. Parmi ces abstractions : les connexions réseau, l'allocation de la mémoire ou l'ordonnancement des tâches.
- **Le composant de configuration :** il fournit des outils génériques pour créer dynamiquement de nouvelles instances de composants. Il permet également de décrire une instance spécifique de la plate-forme comme un assemblage de composants.

Les sections suivantes présentent avec plus de détails ces différents composants.

3.5.1 Le composant de liaison

La notion de liaison dans Jonathan est basée sur celle de RM-ODP [48]. La liaison est le processus d'association ou d'interconnection d'un ensemble d'objets répartis suivant des sémantiques de communication spécifiques. Elle consiste à établir un chemin d'accès entre les objets par l'intermédiaire, notamment, d'objets de localisation et de connexions réseau.

Un nom est une information qui permet d'identifier un objet. Cette information n'est valide que dans un certain contexte de nommage qui définit un ensemble d'associations entre les noms et les objets. Jonathan utilise deux catégories de noms :

- un identificateur : il désigne l'interface d'un objet dans un contexte de nommage donné ;
- un identificateur de session : il désigne une session dans le contexte d'un protocole de communication.

La forme la plus simple de liaison est l'association d'un identificateur `id` avec l'objet qu'il désigne. Cette association s'établit à travers l'opération `id.bind()` qui renvoie l'objet lié. Cet objet doit être préalablement rendu disponible dans le contexte de nommage `nc` à travers l'opération `nc.export(obj)`.

Des formes plus complexes de liaisons peuvent également avoir lieu. Par exemple, la liaison qui relie un client et un serveur à travers une souche du client, une session de communication et un squelette du serveur. Ces objets liaisons sont créés par des usines de liaisons. Ces usines rassemblent tous les objets nécessaires (protocoles de communication, objets intermédiaires, ressources nécessaire, etc.) à mettre en œuvre une certaine forme de liaison. Le composant de liaison

de Jonathan inclut des interfaces et des classes pour la gestion des noms et des liaisons. Ces interfaces et classes peuvent être enrichies ou spécialisées lors du développement de personnalités spécifiques.

3.5.2 Le composant de communication

Ce composant est basé essentiellement sur deux notions : protocole et session. Un protocole fournit un service de communication pour l'échange de messages à travers le réseau. Les protocoles sont organisés sous forme de pile ou de graphe, chaque protocole utilise les services fournis par les protocoles de niveau inférieur. Cette architecture est inspirée de la plate-forme *xkernel* [42].

Une session représente un canal de communication, utilisé pour transmettre des messages entre des processus différents. Une session possède deux interfaces : `Session_Low` et `Session_High` qui servent respectivement à recevoir et à envoyer les messages. Un protocole représente un gestionnaire de sessions en agissant comme un contexte de nommage et de liaison. Il les instancie et les ferme et leur alloue les ressources nécessaires.

Les sessions sont organisées sous une forme similaire à celle des protocoles. Les messages transmis entre deux entités communicantes parcourent le graphe de sessions de haut en bas (l'entité émettrice) puis de bas en haut (l'entité réceptrice). Chaque session peut introduire des traitements spécifiques sur les messages. Au plus bas niveau de la hiérarchie des sessions, nous trouvons les connecteurs (les objets `connection`) qui encapsulent les primitives des protocoles de transport sur le réseau. Les connecteurs sont également des sessions.

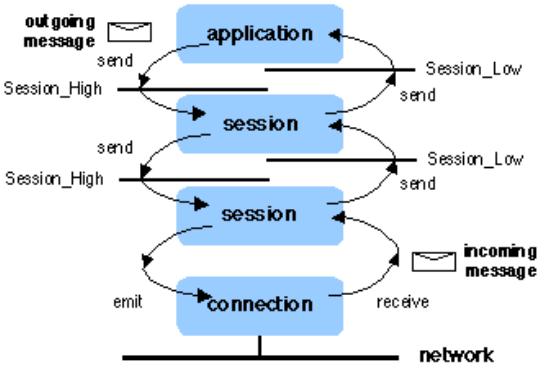


Fig. 3.4 – Exemple d'une pile de sessions dans Jonathan

Les deux principales primitives de communication sont l'envoi et la réception de messages. Ces deux opérations sont différentes, compte tenu de la nature asynchrone de la réception. Une opération de réception bloque le *thread* qui l'exécute jusqu'à ce que des données soient disponibles dans le canal de réception associé à la connexion. Quand un message arrive, le *thread* est débloqué et le message est remis aux couches supérieures en appelant les interfaces `Session_Low` de leurs sessions. Une opération d'envoi se déroule en appelant successivement les interfaces `Session_High` des sessions du graphe jusqu'à la méthode `emit` de l'objet `connection`. Ce mécanisme est décrit par la figure 3.4.

L'établissement d'une session entre des entités communicantes s'effectue à l'aide des opérations `export` et `bind`. Le graphe de protocoles, après sa création au niveau du serveur, exporte une interface de session (`srv_itf`) et obtient un identificateur de session (par exemple, dans le cas de TCP/IP, l'adresse du serveur et un numéro de port). Cet identificateur peut être transmis sur le réseau et utilisé par un client pour se connecter à la session.

Un client se connecte à la session en utilisant la méthode `bind` fournie par l'identificateur de session et en fournissant son interface inférieure comme paramètre (`clt_itf` de type `Session_Low`). La méthode `bind` retourne une interface de type `Session_High` qui peut être utilisé par le client pour envoyer des messages au serveur. Les messages envoyés par le client vers le serveur sont reçus par l'interface `clt_itf`. Ce mécanisme de liaison et d'export est décrit dans la figure 3.5.

Les protocoles de bases implémentés dans Jonathan sont TCP, IP Multicast et une version simplifiée de RTP (*Real Time Protocol*).

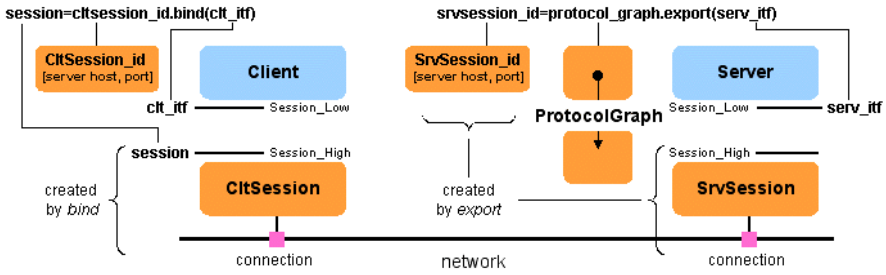


Fig. 3.5 – Etablissement d'une session dans Jonathan

3.5.3 Le composant de configuration

Ce composant permet d'externaliser la structure de la plate-forme Jonathan en une configuration. Il fournit des mécanismes pour la configuration de la plate-forme au moment de l'initialisation. La reconfiguration au moment de l'exécution n'est jusqu'à présent pas prise en charge.

Cette configuration est conforme à une description XML fournie par l'utilisateur. Sa modification entraîne l'altération de la structure de la plate-forme et permet donc de l'adapter à des conditions d'exécution spécifiques . A la base, se trouve un modèle de composition dans lequel les composants forment une configuration spécifiée par un ensemble de propriétés. Chaque instance d'un composant est générée par une usine [29], qui prend comme paramètre le nom du composant et une description des différents paramètres utilisés pour créer une instance spécifique du composant (contexte).

Le composant de configuration inclut donc deux fonctions essentielles :

– La description d'instances spécifiques des composants en utilisant les configurations et les contextes.

– La création dynamique des composants en utilisant les usines.

Les usines

Une usine d'une classe A est une classe qui sert à créer des instances de A. Dans Jonathan, les usines dérivent de la classe abstraite `GenericFactory` qui implémente à son tour l'interface `Factory`. Celle-ci définit une méthode `newObject(Context c)` qui retourne un nouvel objet en utilisant l'information contenue dans le contexte `c`. La classe `GenericFactory` fournit un mécanisme générique, basé sur un ensemble d'objets, et permet aux objets existants d'être réutilisés au besoin.

Contextes et configurations

Un contexte (`context`) est l'abstraction d'un contexte de nommage. Il représente un ensemble d'éléments ou d'objets typés qui sont identifiés par un nom. Le contexte permet d'associer un objet à un nom.

Une configuration est une forme particulière d'un contexte sous la forme d'un arbre. Elle contient quatre types d'éléments : alias, atome, assemblage et propriété :

– un *alias* est simplement un nom alternatif d'un élément ;
– un *atome* désigne la classe qui implémente l'objet ;
– un *assemblage* permet de créer des instances de composants en utilisant une usine et une configuration C. Il décrit un élément qui est composé d'autres éléments. L'usine instancie ces éléments suivant la configuration C. Cette configuration peut décrire à son tour d'autres assemblages qui sont nécessaires pour créer chacun de ces éléments ;
– une *propriété* contient la valeur d'un type primitif.

Ci-dessous l'extrait d'un exemple de configuration de Jonathan.

```
[/]
   [jonathan]
       ...
       [IPv4ConnectionFactory]                        -- une configuration
           factory => org.objectweb.jonathan.libs.resources.tcpip.
                      IPv4ConnectionFactoryFactory    -- un atome
           instance => {factory, .}                   -- un assemblage
           verbose -> /jonathan/tcpip/verbose         -- un alias
       ...
       [JConnectionMgr]
           factory => org.objectweb.jonathan.libs.resources.tcpip.
                      JConnectionMgrFactory
           instance => {factory, .}
           connection_factory -> /jonathan/Ipv4ConnectionFactory/instance]
       ...
       [JDomain]
           [binders]
               0 -> /david/orbs/iiop/instance
```

```
        org.objectweb.david.libs.binding.orbs.iiop.IIOPORB =>
                  (int.class, 0)                -- une propriété
...
[tcpip]
    verbose => (Boolean.class, false)           -- une propriété
...
```

3.5.4 Le composant de ressources

Ce composant regroupe les composants responsables de la gestion des différents types de ressources utilisées par la plate-forme.

- **La gestion de la mémoire :** l'objectif de cette gestion est de réduire la surcharge d'exécution due à des opérations comme la division, la duplication ou l'allocation de données.

- **Les objets Marshallers et Unmarshallers :** les *marshallers* sont utilisés pour convertir les données en une forme standard qui soit adaptée à la transmission sur le réseau (emballage). Les *unmarshallers* effectuent l'opération inverse (déballage). Ces deux objets fournissent des méthodes qui permettent d'écrire et de lire des données de différents types (entiers, réels, chaînes de caractères, etc.).
- **La gestion des activités :** les activités sont représentées par les objets `jobs`. Elles peuvent s'exécuter en parallèle selon des politiques, définies dans des objets `schedulers`, qui prennent en considération les ressources disponibles dans la machine.
- **Le composant de connexions :** les objets `connection` fournissent les mécanismes de bas niveau pour l'établissement des connexions réseau et l'émission et la réception des données sur le réseau. Le composant de connexions est organisé en deux niveaux. Au niveau supérieur, un gestionnaire de connexions permet de gérer et de réutiliser les objets `connection` pour les sessions clientes (par exemple, socket client) et les objets `server_connection_factories` pour les sessions serveurs (par exemple, socket serveur). Si l'objet requis par le client ou le serveur n'est pas disponible, le gestionnaire de connexions instancie un nouveau en faisant appel à l'usine `connection_factory` du niveau inférieur. Par exemple, si le client à déjà ouvert une socket TCP avec le serveur, le gestionnaire de connexions la réutilise et l'affecte à la session supérieure. Dans le cas contraire, il en crée une nouvelle. Cette structure sépare entre les fonctions de gestion des connexions et les fonctions de création de connexion.

3.6 Synthèse

Nous avons abordé dans ce chapitre les intergiciels qui constituent un concept bien implanté dans le domaine des systèmes répartis fixes. Ils fournissent aux développeurs d'applications un haut niveau d'abstraction en occultant la complexité due à la répartition. Les différents types d'intergiciels existants qu'ils soient orientés objets, orientés transaction ou orientés messages, ont pour objectif commun de faire apparaître le système réparti comme une seule entité de traitement

intégrée. Ils incorporent un large ensemble de services allant du simple transport des données jusqu'aux fonctions plus avancées comme les transactions ou la sécurité.

Nous avons étudié la spécification JMS qui constitue actuellement une base commune pour le développement des intergiciels orientés messages MOMs. Ce standard ne constitue pas une innovation majeure dans le domaine des MOMs mais représente plutôt un consensus autour des services existants. JMS est intégré actuellement dans plusieurs serveurs d'applications et occupe souvent le rôle de bus ou d'ossature de communication entre les différentes entités d'un système d'information d'entreprise.

JMS reprend les avantages principaux des MOMs qui sont le faible couplage entre les entités qui communiquent, la fiabilité et la forte disposition au passage à l'échelle. JMS permet également la conception d'architectures ouvertes et évolutives. Grâce notamment au modèle de publication/souscription, il intégre facilement de nouveaux composants dans un système d'information sans effectuer des modifications importantes. En outre, JMS inclut une propriété très intéressante qui est le découplage entre les sémantiques d'envoi de messages et les sémantiques de leur traitement. Cette propriété n'est pas toujours présente dans d'autres types d'intergiciels, comme les intergiciels orientés objets, où le traitement et la communication des requêtes sont entremêlés.

Par ailleurs, nous avons étudié, Jonathan, une plate-forme générique et extensible pour la conception d'intergiciels. Les aspects essentiels de Jonathan, qui nous intéressent dans le cadre de cette thèse, sont les suivants :

- La modularité à travers la séparation entre les différents éléments qui peuvent composer un intergiciel : communication, liaison, ressources, etc.
- La séparation claire entre les abstractions et les implémentations concrètes.

Ces aspects permettent l'introduction d'extensions et de modifications sur ces composants afin de répondre à des besoins spécifiques. Jonathan fournit également des composants de base réutilisables (connexions TCP, emballage de données CDR, etc.), et des composants qui permettent l'application, par exemple, de nouvelles politiques de gestion des ressources ou de communication. Ces éléments sont exploités dans notre plate-forme, comme nous l'exposons dans la deuxième partie de ce document.

Chapitre 4

Les intergiciels mobiles

La différents intergiciels, que nous avons présentés dans le chapitre précédent, ont été conçus avec l'hypothèse que les machines sont puissantes, stationnaires et connectées à des réseaux fixes. Leur utilisation dans les environnements mobiles a révélé inévitablement certaines insuffisances car ils ne prennent pas en considération le caractère très dynamique de ces environnements. D'où un certain besoin d'intergiciels modernes qui prennent en charge les contraintes et également les possibilités qui caractérisent les environnements mobiles. Le rôle de ces intergiciels est d'autant plus important dans les architectures des systèmes mobiles, qu'ils doivent fournir, en plus des fonctionnalités classiques de leurs homologues fixes, une prise en charge de problématiques spécifiques, comme la variabilité de la qualité de service du réseau, la mobilité des utilisateurs ou les ressources limitées des machines. Ces intergiciels doivent également faire face parfois à des topologies réseau complètement dynamiques et décentralisées.

Le principe de l'abstraction, qui a longtemps guidé la conception des intergiciels fixes, n'est plus appliqué d'une manière aussi dogmatique dans les environnements mobiles. Comme nous allons le constater, au long de ce chapitre, les intergiciels mobiles doivent fournir aux applications une vue de certains détails de bas niveau. Ces détails permettent aux applications de prendre des décisions et de modifier le comportement de l'intergiciel suivant le contexte d'exécution.

L'étude que nous présentons dans ce chapitre fournit un point de vue général des technologies les plus significatives, en mettant l'accent, non seulement sur les solutions, mais aussi sur les objectifs qui restent à atteindre. Nous classons, dans une première partie, les intergiciels mobiles selon le modèle de répartition qu'ils adoptent. D'autres intergiciels, se distinguant plutôt par leur architecture, sont exposés par la suite. Enfin, nous décrivons les services qui sont présents fréquemment dans les intergiciels mobiles.

Cette étude nous a permis également de dégager quelques points clés qui, à notre avis, devraient orienter la conception des intergiciels mobiles. Ces points clés ainsi qu'un comparatif des différents types d'intergiciels font l'objet d'une discussion.

4.1 Les modèles de répartition

Cette section classifie les intergiciels mobiles selon le modèle de répartition qu'ils adoptent (invocation d'objets distants, passage de messages, tuples, etc.).

4.1.1 Les intergiciels orientés objets

Comme nous l'avons évoqué précédemment dans la section 3.2.2, l'apport essentiel des intergiciels orientés objets est de faire la jonction entre les concepts de la programmation orientée objet et des modèles de répartition sophistiqués. Ces intergiciels se sont efforcés de masquer les problèmes de répartition et d'hétérogénéité des plates-formes afin de faciliter le développement des applications. Ils permettent aux développeurs de se focaliser essentiellement sur les fonctionnalités des applications. Ils se caractérisent par une très grande richesse fonctionnelle et bénéficient d'une forte intégration avec les langages de programmation.

L'intérêt principal de l'utilisation de ce type d'intergiciels dans les environnements mobiles est de pouvoir réutiliser le patrimoine applicatif existant et de faire interopérer les terminaux mobiles avec les plates-formes fixes. Beaucoup de travaux [61] ont eu pour objet l'adaptation des intergiciels orientés objets aux environnements mobiles et particulièrement aux environnements nomades. L'objectif visé consiste généralement à permettre aux utilisateurs de changer de localisation ou de réseau tout en restant connectés. Une prise en charge minimale de la déconnexion y est incluse. Il existe également des travaux visant à inclure dans ces intergiciels un paradigme d'interaction semi-asynchrone. Ces intergiciels améliorent le déroulement des invocations distantes en utilisant des tampons pour stocker temporairement les messages lors des déconnexions intermittentes. Parmi ces intergiciels, citons Rover [50] et MobileDCE [89].

Nous présentons ci-dessous quelques travaux visant l'adaptation du standard orienté objet CORBA aux environnements mobiles. CORBA (*Common Object Request Broker Architecture*) [79] est la technologie de référence dans le domaine des intergiciels orientés objets. C'est un ensemble de standards et de concepts définis dans le cadre de l'OMG et destinés aux systèmes répartis ouverts. Dans cette architecture, les méthodes des objets distants sont invoquées de manière transparente à travers un ORB (*Object Request Broker*). L'ORB est responsable, dans le cadre d'une requête, de toutes les opérations qui concernent la recherche de l'implémentation de l'objet, la préparation à la réception de la requête et le transport des données. Les interfaces visibles à un client sont complètement indépendantes de la localisation de l'objet, du langage de son implémentation ou de tout autre aspect qui ne soit pas reflété par l'interface elle-même de l'objet. Les interfaces de CORBA sont définies dans le langage IDL (*Interface Definition Language*). Ce langage définit le type d'un objet suivant les opérations que celui-ci implémente ou suivant les paramètres de ses opérations.

Afin de garantir l'interopérabilité entre les différents ORBs qui adhèrent à CORBA, l'OMG a défini le protocole GIOP (*General Inter-ORB Protocol*). Ce protocole spécifie une syntaxe pour le transfert des messages et un ensemble de types de messages que s'échangent les ORBs entre eux. GIOP est conçu pour évoluer directement sur n'importe quel protocole de transport orienté connexion. Le protocole IIOP (*Internet Inter-ORB Protocol*) est, en quelque sorte, l'instantiation de GIOP sur Internet. Il spécifie la manière dont les messages GIOP sont échangés en utilisant

des connexions TCP/IP.

CORBA n'inclut pas initialement une prise en charge spécifique pour les systèmes embarqués ou mobiles. Les ORBs nécessitent généralement un espace mémoire important et la taille des souches et des squelettes générés est plutôt importante. Afin d'adapter le standard CORBA à ce type d'environnements, l'OMG a défini des sous-ensembles ou des extensions de CORBA : le protocole MIOP [74] qui standardise l'accès aux protocoles de transport Multicast ; Minimum CORBA [75] qui fournit une version légère de CORBA destinée aux terminaux à faibles ressources ; et Wireless CORBA [80] qui concerne la gestion de l'accès des terminaux mobiles à travers des réseaux sans fil.

Le protocole MIOP

MIOP (*Multicast InterORB Protocol*) vise à fournir à CORBA une interface standardisée aux protocoles de transport Multicast. L'emploi de ces protocoles est particulièrement intéressant dans les environnements mobiles, car ils permettent de diffuser des données à plusieurs nœuds tout en préservant la bande passante. Par ailleurs, MIOP permet d'assurer le mécanisme de découverte de services dont l'importance est fondamentale dans ce type d'environnements. Les clients mobiles ne sont généralement pas informés des adresses des services disponibles dans leur voisinage réseau. Pour découvrir les nœuds qui fournissent un service donné, le client MIOP envoie une requête à un groupe Multicast dont l'adresse réseau est bien connue. Les serveurs, qui fournissent ce service, se mettent en écoute au niveau du groupe Multicast et envoient une réponse au client.

Minimum CORBA

Cette spécification est un sous-ensemble de CORBA, destinée aux systèmes possédant des ressources limitées, comme les terminaux embarqués ou mobiles. Elle définit un profil qui conserve les principaux avantages de CORBA : la portabilité des applications et l'interopérabilité entre les ORBs. Minimum CORBA prend entièrement en charge le langage IDL, de manière à ce que toute application CORBA bénéficiant de ressources suffisantes pourra évoluer, sans modification, sur un ORB CORBA aussi bien standard que minimal. L'inconvénient majeur de Minimum CORBA est l'absence des aspects dynamiques de CORBA. Les systèmes qui l'utilisent doivent faire des choix précis au moment de la conception, ce qui peut constituer un certain handicap dans les environnements mobiles.

Wireless CORBA

Cette spécification concerne la gestion transparente et simple du changement de localisation d'un ORB client. L'une des idées de base est qu'un ORB fixe ne doit pas nécessairement implémenter cette spécification pour interagir avec les ORBs mobiles.

L'architecture Wireless CORBA (figure 4.1) identifie trois domaines différents qui contiennent

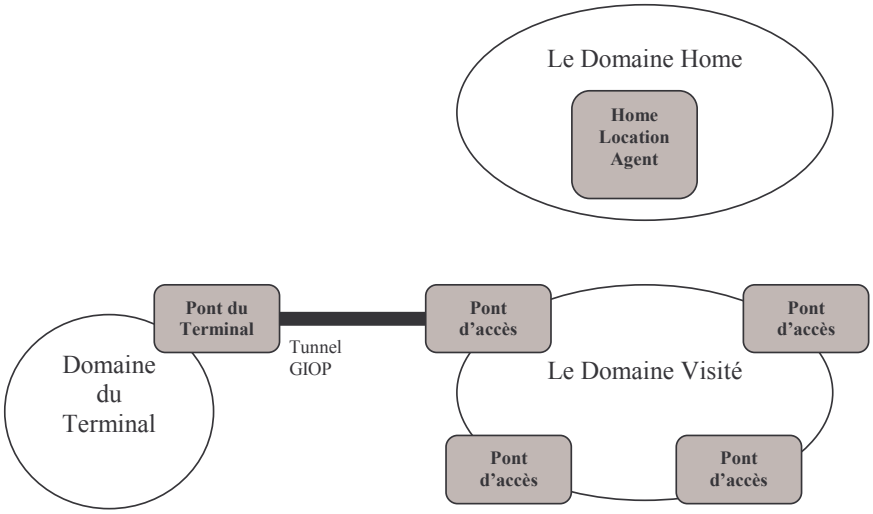


Fig. 4.1 – Architecture de la mobilité des terminaux dans CORBA

chacun un composant central. Le domaine *home* est le domaine de base du terminal mobile. Il contient le composant HLA (*Home Location Agent*) qui est responsable de suivre la trace de tous les terminaux appartenant à ce domaine. Le HLA fournit des opérations pour découvrir et mettre à jour les positions des terminaux. Le domaine visité contient un ou plusieurs ponts d'accès permettant l'accès des terminaux mobiles. Le domaine du terminal consiste en un terminal, qui héberge un ORB, et un pont du terminal à travers lequel les objets CORBA du terminal peuvent communiquer avec les objets distants. Ce pont est chargé d'intercepter toutes les invocations reçues ou initiées par les objets CORBA du terminal mobile.

Les messages envoyés et reçus par un terminal mobile passent par un tunnel GIOP établi entre le pont du terminal et le pont d'accès du domaine visité. Ce tunnel utilise le protocole générique GTP (*GIOP tunneling protocol*) qui définit la manière d'envoyer les messages GIOP. GTP est un protocole abstrait et indépendant du protocole de transport sous-jacent. Lors d'une connexion GIOP, entre un objet sur le terminal et un objet sur le réseau, le pont d'accès est responsable de la translation entre le protocole IIOP utilisé par l'objet sur le réseau et le protocole GTP utilisé par le terminal.

Wireless CORBA définit également la notion de référence relogeable ou IOR Mobile (*Interoperable Object Reference*). L'IOR mobile permet d'identifier le pont d'accès et le terminal sur lequel réside l'objet cible. En outre, elle identifie le HLA qui garde la trace du pont d'accès auquel est rattaché le terminal mobile actuellement. Si un objet CORBA est instancié par un certain terminal mobile, les invocations qui lui sont adressées doivent être redirigées vers le pont d'accès actuel de ce terminal. Ainsi, le profil IIOP de l'IOR mobile ne contient pas l'adresse IP et le numéro de port de l'objet sur le terminal, mais plutôt l'adresse et le numéro de port du HLA du terminal.

La figure 4.2 représente la manière dont les invocations CORBA sont adressées à un objet résidant sur le réseau fixe. La connexion GIOP est divisée en deux parties : une partie entre

le terminal (PT) et le pont d'accès (PA), et une partie entre le pont d'accès et l'objet dans le réseau (S). L'invocation, entre un client CORBA sur le terminal mobile et un serveur CORBA se déroule de la manière suivante : le pont du terminal encapsule l'invocation GIOP dans un message GTP et l'envoie au pont d'accès (1). Le pont d'accès décapsule le message GIOP et effectue une invocation CORBA ordinaire sur l'objet serveur (2). Une réponse éventuelle du serveur arrive au pont d'accès (3), qui l'encapsule dans un message GTP, renvoyé ensuite au pont du terminal (4). Enfin, le pont du terminal délivre la réponse au client.

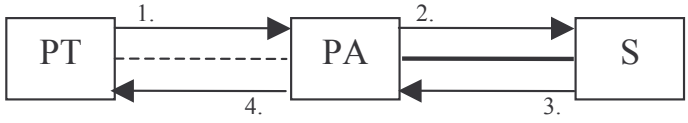


Fig. 4.2 – Invocation d'un objet sur le réseau fixe

L'invocation d'un objet qui réside sur un terminal mobile rattaché à un HLA (figure 4.3) est légèrement plus complexe. Dans le cas où l'IOR de l'objet, appartenant au profil IIOP, contient l'adresse du HLA, le client invoque d'abord le HLA (1). Le HLA détermine le pont d'accès actuel du terminal et remplace l'adresse, contenue dans le profil IIOP reçu, par l'adresse du pont d'accès. Il envoie ensuite cette nouvelle adresse au client (2). A la réception de ce message, le client invoque cette fois le pont d'accès (3). L'invocation se déroule par la suite d'une manière similaire à celle d'un objet sur le réseau fixe (4, 5, 6).

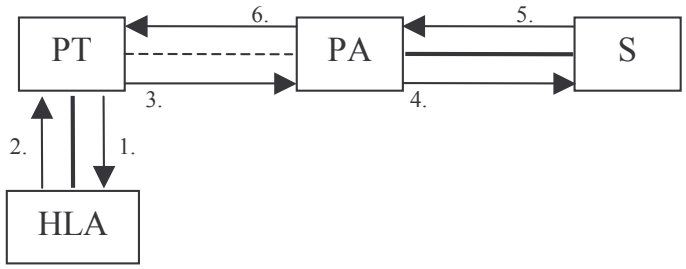


Fig. 4.3 – Invocation d'un objet sur un terminal mobile

Mobiware

Mobiware [2] est un intergiciel, basé sur CORBA, qui vise à adapter la qualité de service fournie par rapport au contexte de l'utilisateur mobile. Comme cela est représenté sur la figure 4.4, les terminaux mobiles sont considérés comme les nœuds finaux du réseau. Les opérations et les services sont développés dans le cœur d'un réseau programmable de routeurs et de commutateurs. Les terminaux sont connectés à des points d'accès et peuvent se déplacer d'un point d'accès à un autre.

L'idée principale de Mobiware est que les terminaux mobiles doivent superviser et s'adapter

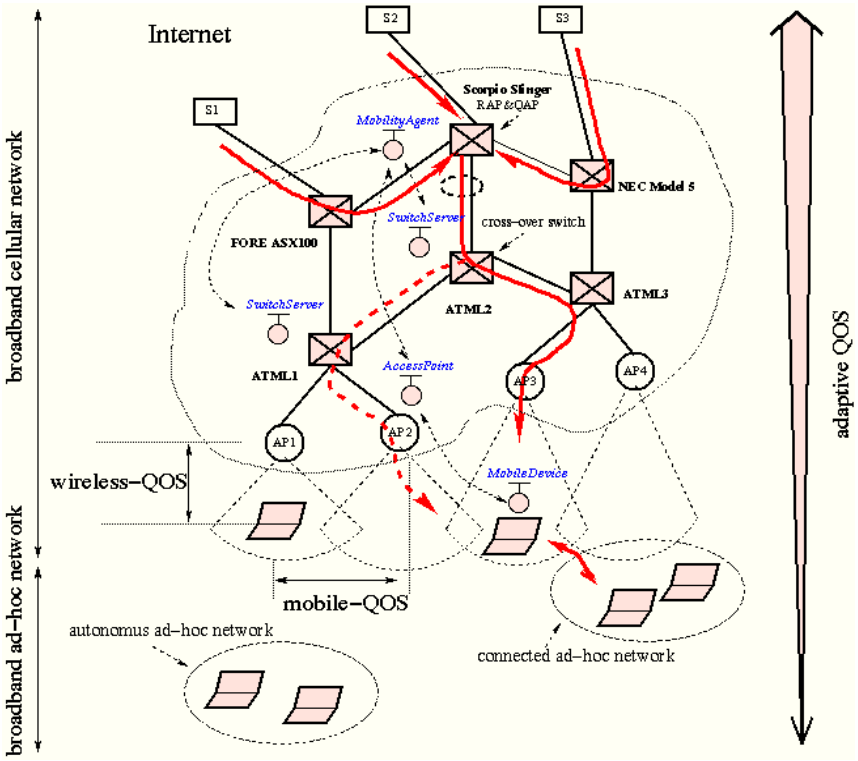


Fig. 4.4 – Architecture de Mobiware

aux variations permanentes des ressources sur le lien sans fil. Le réseau expérimental de Mobiware se compose des entités suivantes : des commutateurs ATM, des points d'accès ATM sans fil et des terminaux mobiles. Deux entités Mobiware peuvent également interagir à travers un réseau ad hoc.

La plate-forme se focalise sur l'adaptation des flux multimédia à différents niveaux de qualité de service. Dans le scénario de base envisagé par Mobiware, les terminaux accèdent aux services situés au cœur du réseau en se déplaçant en permanence, mais en restant toujours connectés. Des fluctuations de bande passante peuvent se produire lors de ces déplacements. Par ailleurs, Mobiware offre une prise en charge additionnelle pour la qualité de service en permettant aux applications multimédia d'opérer de manière transparente pendant les périodes de transfert et de fluctuations persistantes de QoS. Il propose également quelques algorithmes dont l'objet est de transporter les flux adaptables, de réduire les pertes dues au transfert (*handoff*) et d'améliorer l'utilisation des ressources.

Les propriétés intrinsèquement adaptables des flux multimédias sont exploitées dynamiquement par les services mobiles en réponse aux conditions variables des réseaux mobiles. Au niveau de chaque entité du réseau Mobiware, des agents sont chargés d'adapter le flux délivré aux terminaux mobiles suivant les conditions du réseau. Par exemple, lors de la diminution de la bande passante dans une cellule sans fil, l'agent d'adaptation, hébergé par le point d'accès, peut décider de réduire la résolution du flux vidéo transmis aux clients (figure 4.4).

La contribution principale de Mobiware se situe dans l'association assez fine entre, d'un côté, le type et la qualité des données multimédias transportées et, de l'autre, la qualité du réseau. L'architecture reste toutefois singulièrement complexe : des objets CORBA, des proxy d'adaptation qui opèrent au niveau des terminaux mobiles, des points d'accès et des routeurs/commutateurs ayant des fonctions de mobilité. Le déploiement de Mobiware ne peut s'effectuer que sur des réseaux actifs et programmables [99], ce qui réduit sensiblement son champ d'application.

Discussion

Dans les conditions très variables qui caractérisent les environnements mobiles, il est admis que les intergiciels orientés objets ne constituent pas une solution optimale [62]. Ces intergiciels ont été conçus avec l'hypothèse que les terminaux sont puissants, immobiles et connectés à des réseaux fixes. De plus, leur paradigme d'interaction synchrone suppose une connectivité permanente et la disponibilité simultanée de toutes les entités communicantes, ce qui n'est pas toujours le cas dans les environnements mobiles.

Par ailleurs, le scénario globalement prévu par ces intergiciels est l'offre de services à partir de plates-formes fixes dans le réseau vers les utilisateurs mobiles. Dans le cas d'un réseau moins structuré ou dans le cas où les services pourraient être fournis par des terminaux mobiles, il est clair que ces intergiciels sont encore moins adaptés.

4.1.2 Les intergiciels orientés messages

Nous avons défini les intergiciels orientés messages dans la section 3.3. Ces intergiciels représentent un paradigme de communication en pleine émergence et servent de base pour de plus en plus d'applications mobiles. Ils permettent à des entités de s'échanger des informations sans détenir une connaissance précise de l'adresse de destination de la notification (adresse IP, numéro de port). Ils permettent également un fort découplage entre les producteurs et les consommateurs de messages. A travers ces caractéristiques, ils peuvent s'accommoder de manière naturelle à une population dynamique d'entités et à la configuration dynamique des interactions entre elles.

Plusieurs intergiciels orientés messages ont été conçus pour les environnements mobiles. iBus//mobile [92] est une implémentation de JMS destinée aux terminaux mobiles interconnectés à travers un réseau sans fil à grande échelle. Un système iBus//mobile est composé de trois types d'entités : des clients mobiles JMS, une passerelle JMS et un fournisseur JMS. Les clients JMS sont conçus suivant le type du terminal mobile utilisé. Sur les terminaux non programmables, le client JMS communique à l'aide de son mode de communication standard (WAP, SMS, etc.). Sur les terminaux programmables dotés de la plate-forme J2ME [97], le client JMS communique à travers une implémentation légère de l'API JMS. L'élément clé de l'architecture de iBus//mobile est la passerelle JMS. Sa fonction est quelque peu analogue à une station de base qui connecte au réseau tous les terminaux mobiles se trouvant dans une zone géographique donnée. Du point de vue du fournisseur JMS, cette passerelle n'est pas plus qu'un client JMS standard. Du point de vue du client JMS, la passerelle est un nœud où transitent les messages et où s'effectue leur conversion d'un format à un autre.

WebSphere Everyplace [45] est une plate-forme d'IBM qui permet l'accès des utilisateurs mobiles aux données de l'entreprise. Cette plate-forme est conçue comme une extension du serveur d'application WebSphere qui intègre la spécification JMS. Parmi les points intéressants de cette technologie, sa capacité à fournir l'accès à des utilisateurs utilisant des protocoles de communications différents (GRPS, SMS, WAP, Wifi, Bluetooth, etc.). Everyplace propose également des services pour l'adaptation du contenu suivant les capacités et les formats de données pris en charge par les terminaux mobiles. Un gestionnaire de synchronisation permet également aux utilisateurs mobiles d'opérer en mode non connecté et de synchroniser le résultat de leurs activités avec les bases de données du serveur, une fois que la connexion est rétablie.

Pronto [117] est également un intergiciel JMS destiné aux environnements mobiles. Les messages échangés transitent par une passerelle qui utilise des composants dynamiques (*plug-in*) assurant des fonctions comme la compression ou le transcodage sémantique des données. La passerelle peut se situer sur le terminal du client ou sur une autre machine du réseau. Dans ce dernier cas, la communication est basée sur Java/RMI. Afin de gérer les déconnexions et réduire le trafic entre le fournisseur JMS et la passerelle, un mécanisme de cache est mis en œuvre. Si la déconnexion d'un client se produit, la passerelle continue de recevoir les messages. Lors de sa reconnexion, le client se synchronise avec le cache pour récupérer uniquement les dernières mises à jour des données. Pronto propose également un mode décentralisé où les messages sont transmis directement des émetteurs vers les récepteurs (sans la présence d'un fournisseur). Bien que ce mode porte l'opérabilité de cet intergiciel à un environnement diffus, certaines sémantiques de JMS comme la persistance où la garantie d'envoi des messages risquent de ne pas être parfaitement respectées.

Nous décrivons ci-après l'intergiciel STEAM [63] qui fournit un modèle d'événements basé sur la proximité physique et logique.

STEAM

STEAM est un intergiciel basé sur les évènements et conçu pour les applications mobiles qui exploitent la notion de proximité, comme la gestion du trafic. Il s'adresse à des types de scénarios incluant un grand nombre de composants, appelés entités, et représentant des objets réels qui communiquent à travers des réseaux sans fil ad hoc. Plusieurs de ces entités peuvent représenter des objets mobiles, comme les voitures, d'autres entités peuvent représenter des objets fixes, comme les feux de circulation. Ces entités interagissent en utilisant une communication basée sur les évènements afin d'échanger des informations sur l'état actuel du trafic par exemple. Un feu de circulation peut propager aux voitures, qui s'approchent, une notification leur demandant de limiter leur vitesse en raison de l'état de la chaussée. Une ambulance peut diffuser sa localisation aux voitures dans le voisinage afin qu'ils lui cèdent le passage.

STEAM est essentiellement basé sur l'hypothèse que dans ce type de scénarios, les entités sont plus susceptibles de communiquer avec leur voisinage physique. Cela signifie que plus les consommateurs sont proches des producteurs de la notification, plus ils seront vraisemblablement intéressés par son contenu. En conséquence, ces notifications ne sont valides que dans un certain périmètre géographique entourant le producteur. Cette limitation a l'avantage de réduire les retransmissions des messages et ainsi de réduire l'utilisation des ressources des machines et du

réseau. En outre, STEAM, contrairement au service de notification de CORBA [77], ne repose pas sur une infrastructure fixe.

Le modèle événementiel STEAM adopte un modèle événementiel implicite qui permet à une entité consommatrice de s'abonner à un certain type d'évènements sans se baser sur des services qui permettent de localiser les autres entités ou leur mandataires (*proxy*). Il exploite un service de communication basée sur la proximité comme un moyen pour que les entités interagissent. Ce service permet à des composants applicatifs de se découvrir mutuellement lorsqu'ils se trouvent dans la même zone géographique. Cette notion de groupe de proximité recouvre des aspects aussi bien géographiques que fonctionnels.

Le filtrage des évènements STEAM instaure un contrôle resserré des informations diffusées des producteurs vers les consommateurs, à travers l'utilisation des mécanismes de filtrage d'évènements. Trois types de filtres d'évènements sont proposés : le sujet, la proximité et le contenu. Les évènements diffusés consistent en un nom et un ensemble de paramètres typés, le nom représentant le type d'évènements. Un filtre de sujet décrit un type particulier des évènements intéressant les consommateurs. Un filtre de proximité décrit la zone géographique dans laquelle un évènement d'un type donné est valide. Un filtre de contenu filtre les évènements suivant les valeurs de leurs paramètres, en utilisant une expression logique. Les filtres de sujet et de proximité sont appliqués au niveau du producteur, alors que les filtres de contenu sont appliqués au niveau des consommateurs. Cette approche de filtrage a l'avantage de réduire le trafic sur le réseau et de mieux résister au passage à l'échelle en répartissant les tâches entre les producteurs et les consommateurs.

L'approche de STEAM est bien adaptée aux environnements diffus localisés, car elle permet des interactions spontanées basées sur le voisinage physique. Cette approche ne repose pas en outre sur une infrastructure centralisée. La contribution essentielle de STEAM est le mécanisme de filtrage multicritères qui est très efficace dans ce type d'environnements. Ce filtrage permet de prévenir la propagation des messages de manière inutile et de mieux utiliser ainsi la bande passante. Il peut s'effectuer aussi bien chez le consommateur que chez le producteur d'évènements. STEAM ne prévoit pas, en revanche, une gestion de la mobilité permettant la propagation des évènements à grande échelle et de suivre donc les consommateurs dans tous leurs déplacements. Son usage reste principalement destiné aux réseaux ad hoc.

Discussion

L'avantage principal des intergiciels orientés messages est leur mode d'interaction asynchrone. Ce style d'interaction permet un fort découplage entre les entités qui interagissent et permet donc de gérer les problèmes de déconnexion et de mobilité. Le mode de diffusion publication/souscription permet de mieux utiliser également les ressources du réseau. Ces intergiciels répondent ainsi aux besoins des systèmes à large échelle.

Cependant, l'une des limites de ce type d'intergiciels est qu'ils ne s'intègrent pas toujours

très bien avec les concepts de programmation orientée objet. En effet, il existe des incompatibilités fondamentales entre la notion d'objet et celle d'évènement. Les évènements sont considérés comme une collection non typée de données (attribut/valeur), alors les langages de programmation prennent en charge uniquement les objets typés. Les applications basées sur les évènements ne peuvent donc pas exploiter totalement les possibilités de l'orienté objet comme le font les applications CORBA par exemple.

4.1.3 Les intergiciels basés sur les tuples

Le concept de tuple a été introduit par Linda [31] qui se définit comme un langage de coordination pour la programmation concurrente. Un espace de tuples est un espace de mémoire associative partagée qui permet à plusieurs processus de communiquer entre eux. Il se présente sous la forme d'un dépôt de données structurées. Le tuple est l'élément de base de la communication, il est crée et stocké par un processus donné afin qu'il soit accessible, de manière concurrente, aux autres processus. Les tuples sont anonymes dans le sens où leur sélection se fait par correspondance d'un certain nombre d'attributs et non par l'identité de leur auteur. La communication est découplée dans le temps et dans l'espace, car les entités communicantes n'ont pas l'obligation d'être connectées en même temps. La durée de vie des tuples est indépendante du processus qui les a générés. Par ailleurs, l'échange de données ne nécessite pas la connaissance mutuelle de l'adresse de chaque entité communicante, car il s'effectue à travers un espace global partagé.

Ces caractéristiques font des intergiciels basés sur les tuples un prétendant intéressant pour gérer les problèmes des environnements mobiles. Les propriétés de ces environnements, comme le style de communication découplé, favorisent la poursuite des traitements, même en absence de connexions réseau. Il permettent aussi de faire communiquer, de manière opportune, des entités qui sont dynamiques et inconnues préalablement.

Les environnements diffus ne disposent pas toujours d'accès à une infrastructure centalisée. Certains travaux ont donc porté sur la manière de représenter et de rendre global un espace de tuples global, bien qu'il soit physiquement réparti. Les solutions diffèrent dans la manière dont elles abordent le problème. Parmi les plates-formes développées, citons : TSpaces [115], JavaSpaces [114] et Lime [70].

Lime

Lime [70] est un intergiciel qui fournit des mécanismes de haut niveau pour le développement d'applications qui se caractérisent par la mobilité physique (terminal) ou par la mobilité logique (agent). Il est caractérisé par un contexte d'exécution inspiré de Linda. Néanmoins, les mécanismes de communication et de coordination ont été repensés afin d'introduire la notion de mobilité et la capacité de réagir à un état donné du contexte.

Dans le cas d'applications évoluant sur des réseaux ad hoc, il ne peut y avoir un contexte global pour le traitement, compte tenu de l'absence d'accès à une infrastructure centralisée. Lime remplace l'espace de tuples persistant et universellement accessible de Linda par un partage volatile d'espaces de tuples se situant sur chaque terminal mobile. Ces différents espaces constituent,

à travers la coordination, un espace global bien qu'il soit physiquement partagé entre les différents terminaux. Cet espace global ne peut pas être totalement reconstitué à tout moment, cela dépend de la connectivité réseau entre les terminaux.

Au niveau de chaque terminal, l'accès au contexte global se fait à travers une interface de l'espace de tuples (ITS) qui est associée de manière permanente et exclusive à ce terminal. L'ITS du terminal accomplit deux fonctions : elle contient les tuples que met le terminal à la disposition des autres terminaux, et permet d'accéder aux données des autres ITS présents dans la communauté. Ainsi, le contenu de l'espace global créé évolue dynamiquement suivant les unités présentes.

Lors de l'entrée d'un nouveau terminal dans la communauté, le contenu de l'ITS de chacun des autres terminaux est réévalué en tenant compte des données de ce nouveau terminal. Les tuples de l'ITS du nouveau terminal fusionnent avec le contenu actuel de l'espace partagé. Le résultat est accessible à travers l'ITS de tous les terminaux présents. Cette séquence d'opérations est appelée engagement, elle est exécutée de manière atomique. Lors du retrait d'un terminal de la communauté, le contenu de son ITS est supprimé de manière atomique de l'espace partagé de tuples. Cette opération s'appelle le désengagement.

L'espace de données défini par LIME occulte les détails de mobilité et de répartition en gérant automatiquement la localisation des tuples. Néanmoins, l'application a besoin parfois d'avoir un contrôle plus fin sur la portion de l'espace qu'elle souhaite accéder. Ce contrôle est opéré à travers l'introduction de paramètres d'identification dans certaines opérations. Ces paramètres sont exprimés en terme d'identificateur du terminal mobile. Ils déterminent la portée de l'espace partagé qui contient le tuple. LIME introduit également une forme de sensibilité aux contexte d'exécution sous forme d'un espace partagé de tuple particulier appelé *LimeSystem*. Cet espace maintient des informations sur les entités présentes dans la communauté et sur les relations qu'elles entretiennent. Ces informations permettent de réagir face au départ d'un terminal ou de déterminer la cause d'échec de certaines opérations. Par exemple, sans la prise en compte de ces informations dans une opération de lecture, il n'est pas possible de déterminer si la cause de l'échec est le fait que le terminal invoqué ne contienne pas un tuple correspondant, ou si le terminal lui-même ne fait plus partie de la communauté.

Le modèle LIME apporte une solution intéressante pour résoudre les problèmes de communication et coordination dans les environnements diffus. Il permet une gestion transparente de l'échange des informations dans un périmètre local. Il introduit également une certaine forme de sensibilité au contexte, à travers la supervision des entités présentes dans le voisinage. Néanmoins, il manque, dans son stade actuel de développement, de mécanismes de tolérance aux fautes qui permettent, par exemple, de résoudre les incohérences résultant d'une déconnexion qui intervient durant le transfert d'un tuple. La sécurité doit être aussi prise en considération, comme dans TSpaces [115] où des listes de contrôle d'accès sont associées aux espaces de tuples.

Discussion

Les intergiciels basés sur les tuples adoptent également un mode d'interaction asynchrone. Ils exploitent la nature découplée des espaces du tuples en prenant en charge les opérations en

mode déconnecté. Leur point fort est la prise en charge de la fusion d'espace de données qui permet de réunir des objets appartenant à des entités différentes dans un seul espace logique. Les données qui résident sur des nœuds différents, peuvent donc être partagées et accédées de manière transparente. En outre, les espaces de tuples permettent d'introduire des mécanismes de persistance avancés, comme les bases de données, et offrent également des outils évolués pour rechercher et indexer les données (*pattern matching*). Ces intergiciels forment une base solide pour la conception d'applications de coordination dans les environnements mobiles.

4.1.4 Les intergiciels pair à pair

Bolcer [11] décrit un système pair à pair (P2P) comme « Toute relation dans laquelle des hôtes multiples et autonomes interagissent d'égal à égal. Un hôte autonome est utile par lui-même, même en l'absence des autres hôtes. Cette relation de pair à pair implique le fait que des fonctions supplémentaires deviennent disponibles pour un pair comme conséquence de sa collaboration avec les autres pairs ».

Les systèmes P2P ont été développés à la base pour les infrastructures fixes. Ils connaissent actuellement un large succès grâce au partage de fichiers sur Internet. Dans ces systèmes, les utilisateurs hébergent directement les ressources qu'ils veulent partager avec les autres, sans faire recours à leur publication sur un serveur centralisé. Ainsi, les services et les données ne sont pas concentrés sur un seul nœud du réseau, mais chaque pair est responsable d'un sous-ensemble des services disponibles.

Les avantages des systèmes pair à pair sont renforcés lorsque l'environnement visé est diffus. Les interactions pair à pair sont le mode de communication le plus naturel dans ces environnements compte tenu de la nature de la connectivité réseau. Les systèmes P2P mobiles sont nés de la combinaison des terminaux mobiles et des réseaux ad hoc. Un système mobile est un système réparti formé par des hôtes qui changent continuellement de localisation physique et qui établissent entre eux des relations basées sur la proximité. La durée d'interaction entre deux hôtes mobiles peut être très brève.

La principale valeur ajoutée des systèmes P2P mobiles consiste à permettre aux utilisateurs de bénéficier des ressources (données, services, capacité de stockage, puissance de calcul, etc.) disponibles dans le voisinage physique immédiat. Ces systèmes posent de nouveaux défis pour les développeurs de systèmes et d'applications. Ils diffèrent par rapport aux systèmes P2P fixes sur les aspects suivants :

- Les pairs ne sont pas joignables entre eux à tout moment.
- Les liens de communications sont plutôt éphémères : les déconnexions et les reconnexions se produisent de manière fréquente et imprévisible.
- La topologie du système change fréquemment.

Ces aspects impliquent donc la conception d'intergiciels qui en tiennent compte en plus des fonctions qu'ils assurent traditionnellement. Nous décrivons ci-après l'intergiciel Proem [54] qui traite quelques-uns de ces aspects

Proem

Proem [54] est un intergiciel ouvert destiné aux systèmes mobiles diffus, il fournit une solution pour le développement et le déploiement d'applications pair à pair. L'architecture de Proem s'articule autour de quatre types d'entités : le pair, l'utilisateur, l'espace de données et la communauté. Un pair est un terminal mobile autonome prenant part dans une relation avec un autre pair. Un utilisateur est une personne qui possède et utilise un ou plusieurs pairs. L'espace de données est une collection d'objets de données que partagent et gèrent un ensemble de pairs. Une communauté est un ensemble d'entités (pairs, utilisateurs, espace de données).

Au niveau de chaque communauté, certains droits d'accès et fonctionnalités sont définis. La communauté représente un domaine de confiance. Afin de prouver son appartenance à la communauté, un pair doit fournir un certificat cryptographique signé par une entité habilitée à le faire. Les entités sont identifiées par des noms qui sont exprimés sous la forme d'URI. Proem fournit un moyen supplémentaire pour référencer les entités en définissant des profils. Un profil est une structure de données, basée sur XML, décrivant une entité. Les profils sont également utilisés pour annoncer la présence à travers le réseau d'entités possédant certains attributs.

Proem comporte un ensemble de protocoles de communication qui définissent la syntaxe et la sémantique des messages que les pairs peuvent s'échanger. Il définit un protocole de transport de bas niveau et trois protocoles de haut niveau. Le protocole de transport est asynchrone, les données sont passées d'un pair à un autre d'une manière atomique. Il utilise XML pour la représentation des messages et peut être implémenté au-dessus d'une variété de protocoles existants comme TCP/IP, UDP ou HTTP. Les autres protocoles sont :

- *le protocole de présence* : il concerne l'échange de messages permettant aux pairs d'annoncer leur présence et leur disponibilité à travers le réseau ;
- *le protocole de données* : il concerne l'échange de messages permettant aux pairs d'échanger et de synchroniser leurs données ;
- *le protocole de communauté* : il concerne l'échange de messages permettant de garantir, d'appliquer et de vérifier les relations d'appartenance à une communauté.

L'environnement applicatif de Proem est composé du kit du développement des *peerlets* et du système d'exécution.

- *Peerlet* : l'environnement de programmation est basé sur la notion de *peerlet*. Les *peerlets* sont des applications pair à pair simples qui adoptent un modèle de programmation évènementielle. Elles sont hébergées par un exécutif qui est responsable de leur instantiation, leur exécution et leur terminaison. L'exécutif envoie des messages aux *peerlets* comme réaction aux changements qui se produisent dans son état interne ou par réaction aux messages reçus des autres pairs.
- *Le système d'exécution* : il permet à un pair de communiquer avec les protocoles de Proem. Il est constitué de trois composantes :
  - la pile de protocoles ;
  - l'exécutif des *peerlets* ;
  - un ensemble de services qui permettent aux *peerlets* d'accéder à des fonctionnalités utilisées couramment (nommage, gestion de données, l'enregistrement à certains types d'évé-

nements, etc.).

Proem fournit un environnement homogène pour le développement et le déploiement d'applications P2P mobiles. Il fournit plusieurs outils prenant en charge ces applications : API, environnement d'exécution, protocoles de communication.

Comme pour les intergiciels orientés messages, le mode d'interaction adopté est le mode asynchrone. Proem fournit un protocole pour la découverte de service, mais ne spécifie pas la manière d'accéder aux services. Le partage de données entre les différents pairs n'est également pas défini.

Proem met en œuvre également le concept de communauté qui est très répandu dans les systèmes pair à pair. La communauté dans ce cas n'est pas constituée que par le voisinage physique mais aussi par une relation de confiance.

Discussion

L'avantage principal des intergiciels pair à pair est leur architecture qui s'accommode parfaitement avec la nature des environnements diffus. Ils permettent de créer des interactions de manière spontanée et permettent le partage des ressources et des données entre les utilisateurs mobiles. Ces intergiciels devront constituer, dans le futur, le socle des applications d'échange et d'interaction entre les utilisateurs mobiles.

4.2 Les architectures

Cette section classifie les intergiciels mobiles selon l'architecture qu'ils adoptent (réflexive et à base de composants).

4.2.1 Les intergiciels réflexifs

La réflexivité peut être définie comme un principe qui permet à un programme d'accéder, de raisonner et d'altérer sa propre représentation. Nous retiendrons la définition formelle suivante [91] : « La réflexivité est la capacité totale d'un processus de se représenter et de se manipuler lui-même de la même façon qu'il représente et manipule les entités de son domaine d'intérêt ». Avant d'être utilisée dans le domaine des systèmes répartis, la réflexion a été introduite dans certains langages de programmation, puis appliquée dans le domaine des systèmes d'exploitation.

La réflexivité utilise généralement deux mécanismes : la réification et la réflexion. La reification expose le comportement interne d'un système (niveau de base), ce qui rend possible l'ajout de nouveaux comportements pour répondre à de nouvelles données dans le contexte ou à de nouveaux types d'applications. Cette exposition s'effectue à l'aide d'une meta interface. La réflexion (adaptation) vise à changer dynamiquement le comportement interne du système en modifiant ou en rajoutant des propriétés (niveau meta). Concrètement, cela implique dans le

cas d'un système mobile, que l'intergiciel est fourni avec un ensemble minimal configurable de fonctionnalités installées sur le terminal mobile et que l'application adapte le comportement de l'intergiciel selon ses propres besoins.

Les mécanismes de réflexivité sont fort intéressants dans la gestion de l'information contextuelle. A travers la réification, l'intergiciel remonte ces informations aux applications, et, à travers la réflexion, les applications peuvent modifier le comportement de l'intergiciel en conséquence.

Le principe de réflexivité a été utilisé également pour mettre en œuvre la reconfiguration. Les options de configuration des intergiciels bénéficient, en effet, de la réflexivité. Les implémentations existantes permettent, en général, de modifier les politiques de concurrence, les stratégies d'emballage de données ainsi que d'autres politiques. La meta interface qu'exporte l'intergiciel réflexif peut permettre également l'ajout ou la suppression de certains types de catégories et de réorganiser la structure interne de l'intergiciel. La reconfiguration de l'intergiciel rend ainsi possible la suppression d'une fonctionnalité lorsqu'elle n'est plus nécessaire et, au besoin, l'ajout de fonctionnalités qui n'étaient pas prévues initialement.

La majorité des intergiciels réflexifs développés, jusqu'à présent, sont des extensions des ORBs CORBA. Parmi eux citons : dynamicTAO [52], OpenCORBA [59], OpenORB [9] et UIC [102]. En revanche, l'intergiciel CARISMA [62] exploite la réflexivité pour maintenir des informations sur le contexte d'exécution. Nous le décrivons ci-après.

CARISMA

CARISMA [62] est un intergiciel qui exploite la réflexivité pour établir des interactions, basées sur la sensibilité au contexte, entre les applications mobiles. Dans ce modèle, l'intergiciel est chargé de maintenir une représentation valide du contexte d'exécution en interagissant avec le système d'exploitation sous-jacent. Le contexte est une notion qui comprend tous les paramètres extérieurs qui peuvent influer sur le comportement de l'application : mémoire, batterie, puissance de calcul, connexion réseau, localisation, etc.

L'application ou le service peut solliciter que le type et la qualité des données transportées soient compatibles avec son contexte d'exécution. Par exemple, un service de messagerie peut souhaiter l'envoi de texte « plein », lorsque la bande passante est élevée, et l'envoi de texte compressé lorsque la bande passante est faible.

Afin de faciliter le développement d'applications sensibles au contexte, CARISMA considère l'intergiciel comme un fournisseur de services configurables. En particulier, le comportement de l'intergiciel par rapport à une application spécifique est décrit comme un ensemble d'associations entre les services que l'intergiciel peut configurer, les politiques qui peuvent être appliquées pour délivrer le service et les différentes configurations du contexte d'exécution. Dans l'exemple précédent, une association est définie entre le service de messagerie, la politique « message plein » et un contexte où la ressource « bande passante » est élevée. Une autre association est, par ailleurs, définie entre le service de messagerie, la politique « message compressé » et un contexte où la bande passante est faible.

La réflexivité est exploitée dans CARISMA dans le but de permettre la gestion du contexte

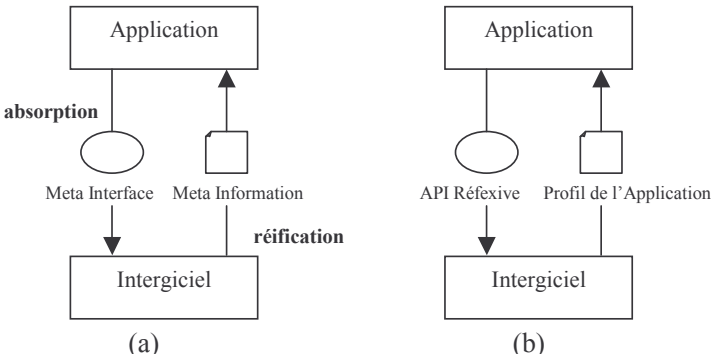


Fig. 4.5 – Processus de réflexion dans CARISMA

et la coopération entre les applications et l'intergiciel. Cette coopération est nécessaire pour trouver la meilleure solution à adopter face à une certaine configuration du contexte d'exécution. Le comportement de l'intergiciel par rapport à une application donnée est réifié dans ce qu'on appelle un profil d'application (figure 4.5). Lors de l'invocation d'un service, l'intergiciel récupère l'état des ressources, que spécifie l'application dans son profil, et détermine ensuite la politique qui doit être appliquée dans le contexte courant.

En raison des changements fréquents du contexte d'exécution et des besoins de l'utilisateur, les concepteurs d'applications ne peuvent pas prévoir toutes les situations possibles. A travers une API réflexive, les applications peuvent inspecter dynamiquement le contenu de leur profil (la configuration actuelle) et l'altérer en ajoutant, en supprimant ou en mettant à jour les associations établies précédemment. Le comportement de l'intergiciel est dicté par les associations formulées dans les profils d'application, le changement de ces informations affecte dynamiquement le comportement de l'intergiciel. Par exemple, une application peut rajouter une association à son profil, afin de demander l'exécution du service de messagerie en utilisant la politique « message crypté ». Cette association permet de protéger les données transmises lorsque la batterie et la bande passante disponibles sont élevées.

Cependant, nous pensons que les associations établies entre les évènements du contexte et les actions de l'intergiciel restent relativement statiques dans CARISMA. Bien que la réflexivité permette la modification de ces associations, le comportement de l'intergiciel reste souvent dépendant de règles prévues à l'avance. L'action que doit entreprendre l'intergiciel doit être plutôt le résultat d'une évaluation dynamique. Par conséquent, les profils des applications doivent plutôt exprimer des besoins que des actions à déclencher.

Discussion

La réflexivité fournit une approche solide et élégante pour la conception d'intergiciels mobiles. En particulier, l'architecture adopte une philosophie de boite blanche qui fournit une représentation compréhensible des détails internes. La réflexivité offre naturellement un haut niveau de configurabilité et de reconfigurabilité. Dans le cas de l'orienté objet, cette configurabilité peut

aller de la modification de l'objet à travers son interface, jusqu'à l'ajout ou la modification des méthodes de l'objet lui-même. La réflexivité permet également d'améliorer la sensibilité au contexte. Un intergiciel réflexif peut fournir, de manière naturelle, des informations contextuelles aux applications qui peuvent ainsi l'exploiter [14].

Malgré ces avantages, les intergiciels réflexifs développés, jusqu'à présent, ne sont pas encore totalement adaptés aux réseaux sans fil et aux terminaux à faibles ressources. Ils adoptent pour la plupart un paradigme de communication synchrone et demeurent relativement lourds en matière de consommation de ressources et d'occupation d'espace.

4.2.2 Les intergiciels à base de composants

Il existe un intérêt grandissant pour l'approche composant dans les systèmes répartis. Selon Szyperski [98], un composant peut être défini comme « une unité de composition avec des interfaces spécifiées contractuellement et des dépendances seulement explicites ». Il rajoute qu'un composant logiciel peut être déployé indépendamment et peut faire l'objet de composition avec une tierce partie. Les technologies composant se basent sur la composition plutôt que sur l'héritage. Pour permettre cette composition tierce partie, elles utilisent des contrats explicites en terme d'interfaces exportées et importées. L'objectif principal est de réduire le cycle de développement des applications, en prônant le développement par assemblage plutôt que le développement à partir de zéro.

Les modèles à base de composants sont des architectures réutilisables qui encapsulent des contraintes et des stratégies pour la composition. Leur apport principal est qu'ils permettent d'appliquer les invariants et les propriétés architecturales visées en contraignant l'interaction entre les composants. De plus, ils simplifient le développement, à travers la réutilisation, et améliorent la maintenabilité et la compréhensibilité du système. Parmi les modèles de composants, nous pouvons citer CCM de CORBA [76], DCOM de Microsoft [66], EJB de Sun [96] et Fractal [12] d'ObjectWeb.

Appliqués aux intergiciels, les modèles à base de composants permettent d'introduire plus de modularité et d'extensibilité à l'architecture, ce qui est un avantage certain dans les environnements mobiles caractérisés par une forte hétérogénéité. Le fait que les différentes fonctionnalités de l'intergiciel, comme la communication, l'empaquetage des données ou la gestion des ressources, soient confinées dans des composants, qui n'interagissent qu'à travers des interfaces explicites, permet de les modifier plus facilement. Il suffit de modifier l'implémentation du composant concerné. Certaines plates-formes permettent de faire ces modifications au moment de l'exécution, ce qui permet de faciliter l'adaptation des applications à des contextes différents.

Les plates-formes à base de composants sont souvent employés pour répondre au besoin d'interopérabilité entre les différents types d'intergiciels utilisés dans les environnements mobiles. Ils offrent également une base pour déployer les applications mobiles dans des configurations matérielles différentes. Par exemple, un composant de l'interface homme/machine, possédant plusieurs implémentations, sera déployé suivant les caractéristiques physiques du terminal ciblé. Un autre exemple est le téléchargement ou la migration d'un composant applicatif d'une machine vers une autre.

Enfin, les modèles à base de composants ont été également utilisés pour caractériser et structurer les données échangées dans le contexte mobile. L'adaptation d'une application, face aux changements des conditions de transmission, peut se faire par rapport au fait que les données, qui sont des composants, peuvent être subdivisées à leur tour en d'autres composants.

Ci-après, nous décrivons l'intergiciel ReMMoC [32] qui illustre l'emploi d'intergiciels à base de composants dans les environnements mobiles.

ReMMoC

L'intergiciel ReMMoC vise à faire interopérer des applications mobiles qui utilisent différents paradigmes de communication comme l'invocation des objets distants, la publication/souscription, les espaces de tuples ou les agents mobiles. L'approche proposée est basée sur le langage WSDL (*Web Services Description Language*) [109] qui permet d'atteindre un haut niveau d'abstraction en dissociant les définitions de services, des intergiciels qui les prennent en charge. Ce langage est utilisé par les services Web qui possèdent des propriétés neutres par rapport au type d'intergiciel utilisé.

L'autre aspect de ReMMoC est la reconfiguration sensible au contexte entre plusieurs paradigmes de communication. Il est basé sur le modèle de composants mis en œuvre par OpenCOM [17], qui est un modèle de composants léger, réflexif et conçu à partir d'un sous-ensemble du modèle DCOM [66].

La figure 4.6 illustre l'architecture de ReMMoC qui consiste principalement en deux gestionnaires de composants : un gestionnaire de composants de liaisons (*mapping component*) et un gestionnaire de composants de découverte de services (*service discovery cf*). Le plus haut niveau de l'abstraction de communication est implémenté par le composant ReMMoC qui agit comme un composant de gestion et de reconfiguration des éléments sous-jacents.

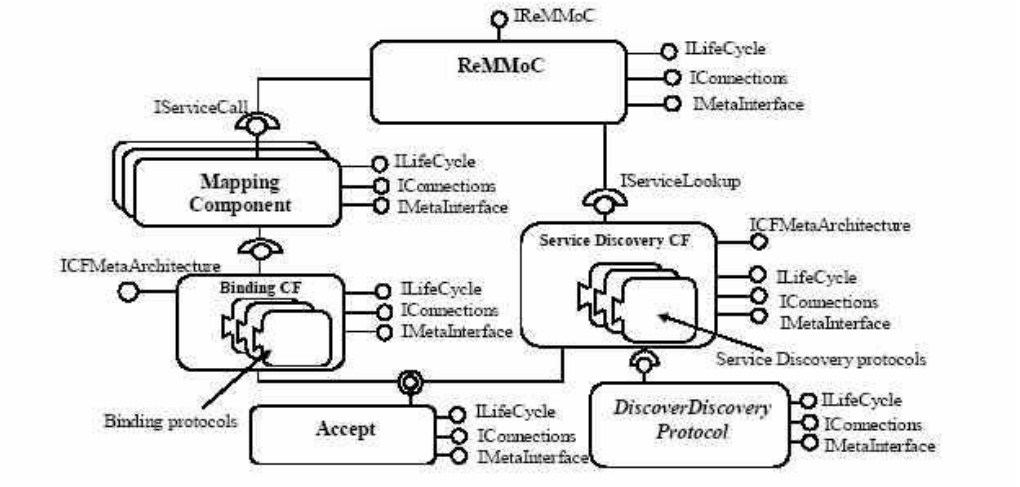


Fig. 4.6 – Architecture de ReMMoC

La fonction principale du gestionnaire de liaisons (MC) est l'interopérabilité avec les entités implémentées sur des intergiciels différents. Suivant la technologie adoptée par ces entités, le MC instancie un composant (*binding cf*) qui permet d'interagir avec eux. Ce composant fournit donc une personnalité spécifique. Par exemple, si l'entité distante est un serveur CORBA, ce composant est configuré comme un client CORBA émettant des requêtes IIOP. Si par contre, l'entité distante est un producteur envoyant des évènements de manière asynchrone, ce composant est configuré comme un consommateur en attente de notifications. Différents paradigmes de communication peuvent être intégrés au gestionnaire de liaison à condition qu'ils soient implémentés en utilisant le modèle de composants OpenCOM. La structure d'OpenCOM gère la configuration et la reconfiguration de ces interactions (liaisons) et garantit que le type de liaison est correct avant que l'opération ne puisse s'effectuer.

Le gestionnaire de découverte de services permet de trouver des annonces de services faites par des protocoles différents. Le gestionnaire est configuré à un moment suivant la technologie utilisée actuellement dans l'environnement. Par exemple, si les annonces de services sont effectuées par le protocole SLP, ce gestionnaire configure et instancie un composant qui se charge de la recherche (*lookup*) SLP. Si, par contre, des annonces SLP et UPNP (voir section 4.7.3) sont découvertes, le gestionnaire configure des composants qui permettent d'interagir avec les deux types d'annonces.

Quel que soit le type d'interaction ou de découverte de services, les applications font appel aux services de ReMMoC en utilisant les mêmes interfaces, respectivement *IServiceCall* et *IServiceLookup* (figure 4.6).

Nous pouvons conclure que REMMOC vise essentiellement à fournir une réponse au problème d'interopérabilité et d'incompatibilité des technologies intergicielles dans les environnements mobiles. En particulier, il propose une vision globale et unifiée des modes d'interaction et de découverte de services. L'utilisation d'une plate-forme de composants garantit la validité et le respect des contraintes par les composants utilisés.

Les tests de performance ont montré que l'exécution d'une personnalité unique sur les terminaux mobiles, de type assistant personnel, donne des résultats acceptables. En revanche, ces terminaux ne peuvent pas prendre en charge la gestion de toutes les personnalités qui peuvent être nécessaires. Par conséquent, cette approche doit être complétée par des mécanismes permettant de télécharger dynamiquement de nouveaux composants.

Discussion

Les plates-formes de composants assurent un bon niveau de configurabilité en fournissant les moyens d'interchanger les composants et de vérifier les dépendances qui existent entre eux. Cette architecture améliore les possibilités d'évolution et d'extension des intergiciels. La reconfiguration des plates-formes de composants n'est pas un sujet totalement maîtrisé. Il existe néanmoins des travaux concernant le développement de modèle et d'algorithmes de reconfiguration qui appliquent de règles de cohérence tout en introduisant le moins de perturbations possibles sur le système [55]. Par ailleurs, l'adaptabilité peut être gérée de manière rigoureuse car les modèles de composants ont le potentiel d'imposer des contraintes appropriées aux processus d'adaptation. La faiblesse essentielle de ces modèles reste leur relative lourdeur et le surcoût de consommation

de ressources qu'ils entraînent en contre-partie de leur souplesse et leur extensibilité.

La réflexivité et les modèles de composants peuvent être des techniques complémentaires. En plus de l'ouverture que fournit la réflexivité, les modèles de composants fournissent de puissants mécanismes de structuration. Leur association permet donc de fournir un haut niveau d'abstraction qui encapsule plusieurs technologies et qui améliore donc l'interopérabilité des intergiciels.

4.3 Services spécifiques dans les intergiciels mobiles

La plupart des intergiciels cités précédemment se caractérisent par leur modèle de répartition ou leur architecture. Dans cette section, nous présentons quelques intergiciels qui se distinguent plutôt par les services qu'ils proposent aux applications. Ces services sont de différentes natures. Certains services existaient déjà dans les environnements fixes, mais se sont révélés très importants dans les environnements mobiles, comme les services de partages de données ou les services de découvertes de services, qui permettent aux utilisateurs mobiles d'exploiter leur capacité de déplacement en accédant à différentes sources de données et de ressources. D'autres services sont spécifiques aux environnements mobiles, comme les services basés sur la localisation qui fournissent aux utilisateurs mobiles des informations dont le contenu dépend de leurs coordonnées géographiques.

4.3.1 Services de partage de données

Nous avons présenté, dans la section 2.3.3, le partage et la réplication de données comme l'une des techniques d'adaptation souvent utilisées. Dans cette section nous traitons les services de partage de données qui ont été intégrés à des intergiciels mobiles. Ces services visent à permettre aux utilisateurs mobiles, qui ont une connexion intermittente, à maximiser leur accès aux données en utilisant la réplication. La réplication dans les environnements mobiles est un compromis entre la cohérence et la disponibilité. La cohérence se réfère à la validité des sémantiques de données.

Ci-après, nous décrivons XMIDDLE [62] qui est un intergiciel de partage de données destiné aux environnements diffus.

XMIDDLE

XMIDDLE [62] permet à des terminaux mobiles, dans le cadre de réseaux ad hoc, de partager et de communiquer des données organisées sous forme de documents XML. Les terminaux peuvent partager les données lorsqu'ils sont interconnectés ou les répliquer et poursuivre le traitement en mode non connecté. XMIDDLE adopte la représentation XML car elle offre une forte structuration. Les données sont organisées sous forme hiérarchique contrairement, par exemple, aux espaces de tuples qui adoptent une organisation plate. Les arbres permettent des manipulations sophistiquées en raison des hiérarchies définies entre les nœuds de données. La structure est typée et les types sont définis à l'aide de schémas XML. Les applications utilisent les analyseurs XML pour valider la structure de l'arbre.

Chaque terminal possède son propre arbre XML qu'il partage avec les autres terminaux. Pour que ce partage soit possible, un terminal importe (monte) explicitement l'arbre d'un autre terminal. Les applications peuvent modifier les arbres en utilisant les opérations définies dans l'API DOM [107]. Ces opérations permettent de parcourir l'arbre et d'ajouter ou de supprimer des nœuds. Chaque terminal possède le contrôle total sur ses propres arbres, mais il doit notifier les autres hôtes connectés, qui ont importé une partie ou la totalité de son arbre, lorsque des modifications se produisent.

Les différents réplicas peuvent évoluer indépendamment les uns des autres lors des périodes de déconnexion. Lors de la reconnexion, les hôtes qui partagent des réplicas en commun doivent les synchroniser. Un mécanisme de réconciliation, basé sur des techniques de différenciation d'arbre, est utilisé à ce propos. Cette approche de réconciliation opère à deux niveaux. Le premier niveau est indépendant de l'application, il est basé sur des techniques de fusion et de comparaison d'arbres XML. Le deuxième niveau spécifie les paramètres de réconciliation d'une manière propre à l'application. XMIDDLE fournit aux développeurs d'applications des primitives pour résoudre les conflits. Le développeur spécifie dans le schéma XML, associé à l'arbre, les primitives à appliquer lorsque des conflits se produisent.

L'architecture de XMIDDLE se base sur le modèle de référence ISO. La couche applicative utilise des primitives, comme *connect*, *disconnect*, *traverse* et *update*, sur son propre arbre DOM ainsi que sur les réplicas distants. La couche de présentation transforme les documents XML en arbres DOM et fournit à la couche applicative les primitives citées auparavant. La couche session gère les connexions et les déconnexions. L'implémentation XMIDDLE de ces couches est instanciée au-dessus de protocoles réseau standards comme TCP ou UDP.

Les limites actuelles de XMIDDLE sont dues à l'inadaptation du paradigme de communication, utilisé actuellement pour le partage des données, qui demeure synchrone. Il est donc vulnérable aux défaillances dues à une déconnexion impromptue ou à la mobilité des utilisateurs.

4.3.2 Services de localisation (géodépendants)

La localisation est l'un des aspects les plus étudiés de la sensibilité au contexte. Les services basés sur la localisation exploitent la possibilité qu'ont les terminaux mobiles d'être informés de leur position physique actuelle. Ces services géodépendants correspondent à un besoin naturel des utilisateurs, ils interagissent généralement avec le système d'exploitation afin d'extraire les informations de localisation et de les présenter sous une forme compréhensible à utilisateur. Les applications sont multiples. Par exemple, les systèmes de navigation utilisent les informations sur les positions actuelles et précédentes pour guider les utilisateurs dans des endroits différents. Ils peuvent servir également à assister l'utilisateur dans ses déplacements pour découvrir des curiosités touristiques, trouver facilement un hôtel ou le distributeur de billets le plus proche.

L'une des limites actuelles de ces services est qu'ils ne prennent pas en charge la diversité des systèmes de positionnement. Souvent, plusieurs versions d'un même service doivent être développés pour rester compatibles avec les différentes technologies de positionnement. Parmi les systèmes de positionnement, citons le GPS, l'identification de cellule, les badges actifs, les fréquences radio (par exemple, Zigbee de Phillips [118]) et l'interprétation des images.

Afin de faciliter et d'améliorer le développement des services de localisation, des intergiciels ont été conçus pour fournir une interface commune aux différents systèmes de positionnement. Certains intergiciels actuels construisent un modèle spatial à partir des informations de localisation. Parmi eux, l'intergiciel Nexus [28] que nous exposons ci-dessous.

Nexus

Nexus [28] est une plate-forme générique qui prend en charge les applications sensibles à la localisation dans un contexte mobile. L'objectif principal de cette plate-forme est de fournir un point de vue unifié et homogène, quelle que soit la technologie de positionnement sous-jacente. Nexus fournit la gestion des composants de positionnement et des modèles spatiaux qui représentent le monde réel.

L'architecture de Nexus est formée par quatre composants qui interagissent (figure 4.7).

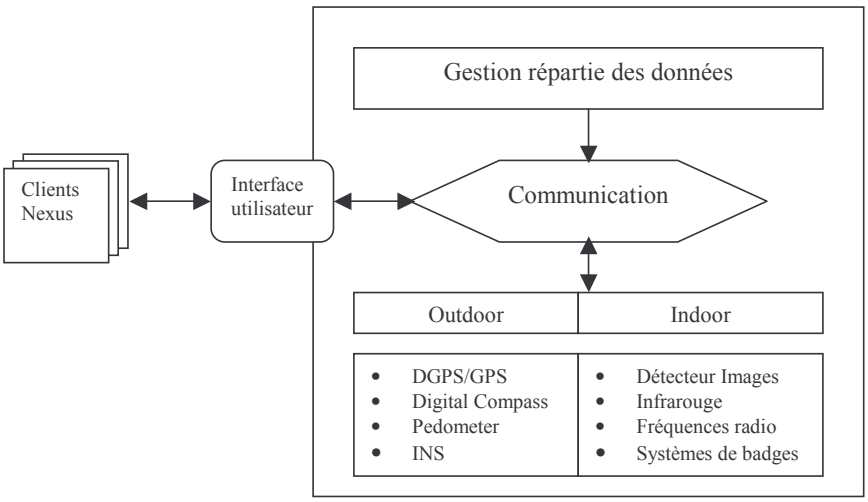


Fig. 4.7 – Architecture de Nexus

Interface de l'utilisateur Cette interface, qui s'exécute sur les terminaux mobiles des utilisateurs, fournit les fonctions de base pour l'interaction avec la plate-forme Nexus et pour l'affichage et la navigation à travers le modèle spatial. Cette interface peut également s'adapter à des terminaux possédant des niveaux de ressource différents en terme de puissance de calcul, de taille d'écran et de type de connectivité réseau.

Environnement de communication Ce composant agit comme un agent intermédiaire qui permet la communication entre tous les autres composants. Nexus peut évoluer au-dessus d'une variété de réseaux sans fil. Dans un contexte global, les réseaux cellulaires comme GSM ou UMTS peuvent être utilisés. Dans un contexte plus restreint, comme à l'intérieur d'un bâtiment,

les réseaux locaux sans fil, comme le 802.11b, peuvent être utilisés. La couche de communication fournit une interface commune à ces différents types de réseaux.

Système de positionnement Nexus distingue deux types de positionnement : le positionnement en plein air (*Outdoor*) et le positionnement à l'intérieur des immeubles (*Indoor*). Les technologies utilisées dans ces deux types de positionnement sont différentes. Pour le positionnent en extérieur, le système le plus utilisé est le GPS, mais il n'est pas utilisable à l'intérieur où les signaux reçus par satellite sont stoppés par les immeubles. Le système d'identification de cellule est également très courant puisqu'il se base sur les infrastructures existantes de la téléphonie cellulaire. Pour le positionnement en intérieur, on utilise des systèmes basés sur l'interprétation d'images ou les badges actifs. Nexus combine plusieurs de ces systèmes pour pouvoir localiser l'utilisateur dans toutes les situations. Il fournit néanmoins aux applications une interface commune leur permettant de faire abstraction des détails de chaque système.

Gestion des données Tous les aspects concernant la gestion des données dans Nexus sont assurés au niveau de ce composant. Les applications basées sur la localisation ont des besoins différents, les données spatiales sont donc fournies avec plusieurs représentations. Des algorithmes appropriés pour déduire tous les niveaux de détails nécessaires sont implémentés sur la plateforme. Pour assurer l'interopérabilité, certaines relations entre les différents modèles sont définies. Le format de données doit être interchangeable. Etant donné que des quantités importantes de données doivent être gérées simultanément, une solution centralisée ne serait pas adéquate. Nexus utilise une architecture répartie pour la gestion des données.

4.3.3 Découverte de services

Dans les intergiciels traditionnels, la découverte de service est assurée en utilisant les noms de services fixes et connus. La stabilité du réseau et des machines rend possible la mise en place d'un service centralisé qui répertorie tous les services disponibles. Les services sont annoncés en publiant un profil contenant leurs attributs, leurs capacités et tout ce qui est nécessaire pour pouvoir interagir avec eux.

Plus la structure du réseau devient dynamique, plus la découverte de services devient problématique. L'approche utilisée dans ce cas est répartie, car il n'existe pas de service centralisé mais plutôt des protocoles qui annoncent la présence de services sur le réseau.

La découverte de service peut généralement se faire de manière centralisée dans les environnements nomades. Ces environnements sont dotés d'un accès à une infrastructure fixe qui peut contenir toutes les informations à propos des services disponibles. La découverte de service adopte donc le même modèle que celui des environnements fixes.

La découverte de service est en revanche plus complexe ou plus coûteuse dans les environnements diffus. En effet, toutes les entités qui sont connectées au réseau peuvent annoncer et fournir des services : le réseau devient un dépôt de services. Le processus de découverte s'appuie sur la diffusion de messages vers les voisins qui peuvent les relayer ensuite pour atteindre les

éléments les plus éloignés. Par conséquent, il y a un compromis à trouver entre la rapidité de la découverte et son coût en terme de consommation de bande passante. Une inondation permet de trouver rapidement un élément sur le réseau, mais elle consomme plus de ressources par rapport à un protocole qui propage un message de manière spontanée. Le processus de découverte doit donc s'appuyer sur un support de diffusion efficace. En outre, il y a également un compromis à trouver entre l'efficacité, en terme de temps de recherche, et l'expressivité des annonces de services.

Le problème de découverte de services concerne plusieurs niveaux de l'architecture d'un système diffus : la découverte d'adresses dans les protocoles d'auto-configuration, la découverte de routes dans les protocoles de routage, la découverte de services de type SLP ou UPnP et la découverte de voisins dans un système pair à pair.

Jini [3] propose une découverte de service basée sur l'utilisation de tables indexant les services disponibles. Ces tables sont gérées par des services particuliers, appelés les services de consultation (*lookup services*). Elles peuvent contenir du code exécutable en plus des informations décrivant le service. Une communauté Jini ne peut pas s'établir sans la présence, au minimum, d'un service de consultation, même si les services et les utilisateurs potentiels résident sur le même hôte physique.

IETF propose le protocole SLP (*Service Location Protocol*) [104] où des agents d'annuaire implémentent les enregistrements concernant les services. Ils stockent les profils et les localisations des services, mais ne stockent pas de code exécutable. La découverte de services nécessite en premier lieu la localisation de ces agents d'annuaires. Si aucun agent n'est disponible, les clients peuvent diffuser en Multicast des requêtes de services et les serveurs peuvent diffuser en Multicast des annonces de leurs services. Les services les plus courants utilisent le modèle de service standardisé par l'IANA (*Internet Assigned Numbering Authority*).

Le projet Salutation [87] utilise également un service relativement centralisé, appelé le gestionnaire SLM. Les clients et les serveurs ne peuvent établir le contact qu'à travers ces SLMs. Les SLMs s'appuient sur divers protocoles de transport. Ils spécifient complètement le format des données transmises, permettant ainsi une relative indépendance par rapport au protocole de transport. Salutation se focalise sur l'interopérabilité de différents types de services à travers un ensemble commun de spécification des fonctionnalités du service.

Nous présentons ci-après le système de découverte de service de l'intergiciel LIME que nous avons décrit de manière générale dans la section 4.1.3.

La découverte de services dans LIME

L'intergiciel LIME propose un mécanisme de découverte de services basé sur la notion d'agent. Un agent est toute entité qui possède la capacité d'exécuter des traitements, il peut représenter des composants applicatifs ou des terminaux.

Chaque service est représenté par un profil stocké dans un tuple. Ce tuple contient l'identificateur du service, une liste d'attributs, une liste de capacités, un objet *proxy* et quelques informations à propos de la communication entre ce *proxy* et le serveur. Quand un service s'en-

registre, le système lui fournit un identificateur unique. Les attributs qualifient les capacités du service. L'objet *proxy* est utilisé par le client pour accéder au service. Cet accès peut être fourni de deux manières : le *proxy* peut implémenter totalement le service (aucune communication distante n'est alors nécessaire) ou il peut fournir une interface à un serveur distant en occultant les détails de communication au client . Notons que le protocole de communication utilisé par le *proxy* et le serveur est arbitraire. Il peut s'agir d'un protocole bien connu (par exemple, Java RMI) ou d'un protocolaire propriétaire spécifique à l'application.

Le *proxy* doit naturellement connaître la localisation du service. Cette localisation peut changer si le service migre à un autre endroit. L'approche adoptée, dans ce cas, consiste à diviser l'information de localisation en deux parties (figure 4.8). La première partie représente la localisation physique de l'agent exécutant le service, la deuxième représente une adresse logique dans l'espace d'adressage de cet agent. Tandis que la mobilité entraîne le changement de la localisation physique, cette adresse logique reste inchangée. Ces deux informations sont donc publiées séparément. La localisation physique est publiée avec l'identificateur de l'agent dans un espace de tuples particulier. Cet espace de tuples contient un tuple de localisation pour chaque agent et son contenu est mis à jour lors de la migration de l'agent. De cette manière, un agent n'a besoin de mettre à jour qu'un seul tuple lors de sa migration indépendamment du nombre de services qu'il fournit.

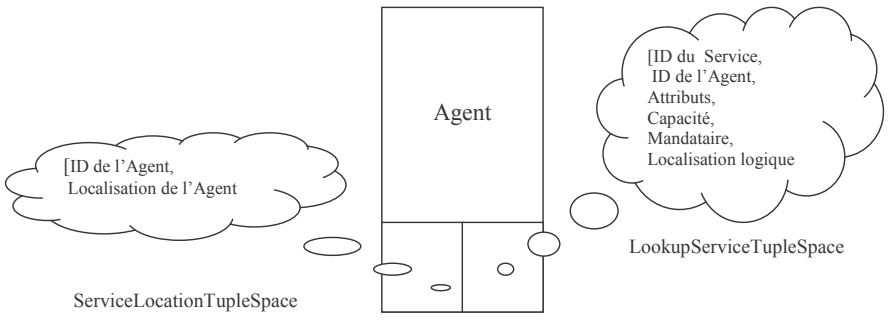


Fig. 4.8 – L'agent et les espaces de tuples locaux dans LIME

Les tuples, décrivant les services qu'un agent souhaite publier, sont stockés dans un espace de tuples local, appelé *LookupTupleSpace*. En utilisant le même nom pour tous les espaces de tuples locaux, le partage transparent des espaces de tuples (voir section 4.1.3) peut être exploité. Par conséquent, chaque espace de publication de service est automatiquement partagé avec les autres agents. Lors de l'engagement d'un nouvel hôte, son espace est partagé avec tous les agents formant la communauté. Etant donné que l 'engagement et le désengagement sont des opérations atomiques, chaque agent possède une vue cohérente des services actuellement disponibles dans le voisinage ad hoc. Durant la migration, le *LookupTupleSpace* d'un agent mobile n'est plus accessible au reste de la communauté.

La découverte de services par un client s'effectue en interrogeant l'espace de tuples de consultation, en utilisant un modèle qui décrit le service recherché. Dans ce modèle, le client peut rechercher des services avec un identificateur spécifique, des services qui possèdent certains attributs, des services qui implémentent certaines interfaces ou une combinaison des critères précédents.

Le partage dynamique de l'espace de tuples permet la mise à jour atomique des enregistrements des services en maintenant leur cohérence à travers les hôtes connectés. Une interface unique (ITS) permet l'accès à l'ensemble de l'espace fédéré comme s'il s'agissait d'un espace local.

4.4 Discussion

L'étude, que nous avons effectué dans ce chapitre, permet de constater qu'il existe quelques propriétés communes que doivent posséder les intergiciels mobiles quel que soit leur type. Ces propriétés leur permettent de mieux s'adapter à la nature des environnements mobiles et de mieux répondre ainsi aux besoins des applications. Elles ont trait à des choix de conception et à des propriétés non fonctionnelles. Nous les décrivons ci-après.

Interaction asynchrone le mode asynchrone s'impose comme le mode d'interaction le plus adapté aux environnements mobiles. Il fournit une solution aux problèmes de déconnexion et de changements d'adresse ou de localisation sur le réseau. Il permet de mieux gérer également la latence importante qui caractérise parfois les réseaux sans fil.

Adaptabilité et sensibilité au contexte L'adaptabilité permet à l'intergiciel d'adopter un comportement efficace face à une situation qui se présente. Elle consiste à avoir plus de souplesse dans le comportement au lieu d'adopter une démarche uniforme et rigide. L'adaptabilité implique généralement toutes les couches du système et suppose une connaissance du contexte dans lequel il évolue.

Le type de sensibilité de contexte le plus répandu est basé sur les informations de localisation. La sensibilité est un concept plus large, son application doit être généralisée de manière à offrir un point de vue global sur l'état d'environnement, comme l'état du terminal, l'état du réseau ou encore les activités de l'utilisateur. La découverte de service peut être considérée comme une forme de sensibilité de contexte car elle permet aux utilisateurs de bénéficier des ressources et des services que leur contexte d'exécution permet d'accéder. L'intergiciel doit maintenir régulièrement des informations sur le contexte d'exécution et agir en conséquence ou, à défaut, permettre aux applications et aux utilisateurs d'exploiter ces informations.

La sensibilité au contexte est souvent liée à l'adaptabilité, car les informations sur le contexte permettent aux applications de solliciter l'intergiciel pour qu'il applique une certaine politique d'adaptation.

Configurabilité et reconfigurabilité La configuration se traduit par la capacité du système d'introduire de nouvelles fonctionnalités, qui n'étaient pas prévues initialement ou de modifier celles qui existaient déjà. La reconfiguration permet de faire ces modifications au moment de l'exécution du système. Elle peut s'effectuer suite à une intervention humaine ou à des évènements qui se produisent dans l'environnement.

Interopérabilité Le besoin d'interopérabilité est encore plus fort pour les intergiciels mobiles, car la mobilité implique le parcours et l'interaction avec des domaines et des environnements différents en terme de ressources et de services. Un intergiciel mobile doit donc avoir la capacité d'interagir avec des services de différents types (orientés objets, basés sur les évènements, espaces de tuples, etc.).

Légèreté Cette caractéristique peut s'opposer parfois aux propriétés citées précédemment. Mais compte tenu des limites en ressources internes des terminaux mobiles, les intergiciels mobiles doivent être optimisés de manière à offrir plus de services en consommant moins de ressources machine. En particulier, la taille de l'intergiciel doit être adaptée au type du terminal, et rester dans des proportions qui n'empêchent pas l'emploi d'autres applications en parallèle. La légèreté peut être améliorée grâce aussi à la modularité et à la configuration en permettant d'assurer uniquement les fonctionnalités dont l'application a besoin.

La liste précédente n'est pas exhaustive, d'autres propriétés comme la tolérance aux fautes ou la sécurité peuvent également être citées. Nous nous somme focalisés sur celles qui sont plus spécifiques aux environnements mobiles. Par ailleurs, il nous paraît difficile qu'un seul type d'intergiciel puisse mettre en œuvre de manière efficace et performante toutes ces propriétés. Leur application dépend du modèle de répartition et du style architectural adopté par l'intergiciel. Chaque style possède ses forces et ses faiblesses, chaque style s'adapte mieux à un certain type d'applications. Le tableau 4.1 situe les types d'intergiciels mobiles, que nous avons cités précédemment, par rapport à ces caractéristiques.

Caractéristiques	**Modèle de répartition**				**Architecture**	
	orienté objet	orienté messages	tuples	P2P	réflexive	composants
Interaction asynchrone		+	+	+		
Configurabilité					+	+
Reconfigurabilité					+	
Adaptabilité et sensibilité au contexte					+	+
Interopérabilité						+
Légèreté		+	+	+	-	-

Tab. 4.1 – Propriétés des différents types d'intergiciels mobiles

Le mode d'interaction dépend du modèle de répartition adoptée. Les intergiciels orientés messages, basés sur les tuples et pairs à pairs adoptent naturellement le mode asychrone (STEAM, LIME, Proem).

L'adaptabilité, la sensibilité au contexte, la configurabilité et l'interopérabilité dependent de l'architecture de l'intergiciel. Les architectures réflexives et basées sur les composants fournissent un cadre pour mettre en œuvre ces propriétés. En particulier, la sensibilité au contexte et l'adaptabilité peuvent être gérées à travers les mécanismes de réification et de réflexion des

intergiciels réflexifs (CARISMA). La réflexivité permet également un haut niveau de configurabilité et de reconfigurabilité. Par ailleurs, les plates-formes de composants présentent une solution aux problèmes d'interopérabilité (ReMMoC).

La légereté dépend de l'architecture mais aussi du modèle de répartition. Les intergiciels orientés messages, basés sur tuples et pair à pair, se caractérisent par une relative légèreté par rapport aux autres types d'intergiciels mobiles. Cette légèreté possède son coût en ce qui concerne les fonctionnalités assurées et les applications se voient donc attribuer plus de responsabilité (typage des données, exécution, activation, etc.). Elle leur assure néanmoins une plus grande efficacité. Par exemple, dans les intergiciels orientés messages, les fonctions les plus coûteuses (transactions, persistance, etc.) sont traitées au niveau du fournisseur qui est hébergé par une plate-forme fixe, ce qui autorise plus de légèreté pour la partie s'exécutant sur les terminaux mobiles. Pour leur part, les intergiciels réflexifs et basés sur les composants demeurent relativement gourmands en matière d'occupation d'espace et d'utilisation des ressources des terminaux.

4.5 Synthèse

Ce chapitre nous a permis de faire un tour d'horizon des différents types d'intergiciels mobiles existants. Cette étude montre les points forts et les faiblesses de chaque type. Elle montre également l'ampleur des problématiques auxquels doivent face les concepteurs des intergiciels mobiles et également la diversité des solutions proposées. Il n'existe pas à l'heure actuelle un seul intergiciel qui peut répondre à la fois aux besoins de toutes les applications et aux contraintes de tous les types d'environnements mobiles.

Les intergiciels mobiles restent donc un vaste domaine de recherche, mais les solutions proposées tendent à converger vers des recommandations et des concepts bien définis que nous avons évoqués au cours de la discussion. Les points les plus importants sont l'adoption d'un modèle de répartition asynchrone et d'une architecture adaptable et configurable. Nous avons retenu cette approche dans le cadre de notre plate-forme, comme nous le décrivons dans le prochain chapitre.

Deuxième partie

Conception et Réalisation

Chapitre 5

MobileJMS : un MOM pour les environnements nomades

Nous avons abordé au cours du chapitre précédent les différentes approches de conception d'intergiciels pour les environnements mobiles. Parmi les différents types d'intergiciels existants, nous voulons faire valoir l'idée que les intergiciels orientés messages MOM constituent une solution intéressante et efficace aux problématiques des environnements mobiles, et particulièrement des environnements nomades. Cette constatation est née du fait que ces intergiciels se distinguent par leur paradigme de communication, leur flexibilité et leur dynamisme, qui sont en phase avec les caractéristiques de ces environnements.

Nous avons conçu et développé l'intergiciel MobileJMS qui est un MOM basé sur la spécification JMS. Ce chapitre est essentiellement consacré à l'étude de cet intergiciel. MobileJMS propose une gestion spécifique, ouverte et extensible des problématiques des environnements nomades. L'originalité de l'approche consiste dans l'adaptabilité et la sensibilité au contexte qui caractérisent la communication entre les entités réparties.

Par rapport aux MOMs existants, MobileJMS se différencie par son mode de communication qui prend en considération les facteurs de déconnexions impromptues, de variabilité de bande passante et de l'accès transparent à travers plusieurs réseaux. La communication simultanée à travers plusieurs protocoles de transport est également l'une des problématiques prises en compte. L'objectif principal est de transporter les messages d'une manière qui soit la plus adaptée au contexte d'exécution courant. Le module de communication de MobileJMS est basé sur les notions de services à valeur ajoutée et de négociation dynamique. L'un de ces services est le service de stockage et ré-émission qui permet aux applications de continuer leur déroulement même en cas d'absence de connexion. Par ailleurs, la flexibilité de la communication dans MobileJMS est également due aux possibilités de configuration et de reconfiguration.

5.1 Motivations

Le choix d'un intergiciel JMS pour prendre en charge les applications dans les environnements nomades s'explique par les raisons suivantes.

Découplage temporel Le mode de communication est totalement asynchrone. Le passage de messages à travers un nœud central permet à des entités de s'échanger des messages même si elles ne sont pas connectées au même moment. C'est souvent le cas dans les environnements mobiles où la connectivité n'est pas permanente. Par exemple, un client peut envoyer une requête à un serveur, se déconnecter, puis recevoir la réponse lors de sa reconnexion.

Découplage spatial Contrairement aux modes d'interaction purement synchrones, les entités peuvent s'échanger des messages sans se référencer explicitement. Les messages ne sont pas envoyés à une adresse précise sur le réseau mais à un emplacement logique. Ainsi, la mobilité des utilisateurs, qui se traduit souvent par le changement de l'adresse physique, n'influe par sur leur capacité à recevoir et à envoyer les messages vers les entités distantes.

Dynamisme et flexibilité Les entités qui communiquent à travers un intergiciel JMS sont anonymes et variables. Les utilisateurs peuvent se joindre et participer dynamiquement à une session et ne sont pas forcément connus à l'avance. Ceci s'apparente assez bien au type de connectivité dans les environnements mobiles où la flexibilité de la connexion permet à des utilisateurs, parfois inconnus, de joindre et de quitter dynamiquement le réseau.

Portabilité Les intergiciels JMS adoptent un modèle de répartition qui offre un point de vue de haut niveau, portable et assez indépendant des mécanismes liés au transport sur le réseau. Les sémantiques d'échange de messages peuvent se baser sur différents modes de transport (synchrone/asynchrone, fiable/non fiable, etc.). Chaque client JMS peut utiliser un mode de transport qui soit différent des modes utilisés par les autres clients. En outre, le découplage, entre les sémantiques du passage de messages et les mécanismes de transport, isole les applications des évènements liés au réseau. Ainsi, une connexion JMS peut être associée à une nouvelle connexion de transport après une déconnexion/reconnexion, ou être associée à deux connexions de transport au même temps.

Il faut noter toutefois que les intergiciels JMS sont plus destinés aux environnements nomades où il existe à un accès à une partie fixe de l'infrastructure. En effet, Le fournisseur JMS (le nœud intermédiaire) est un composant qui nécessite, d'une part, des ressources significatives pour pouvoir évoluer, et d'autre part, une connectivité permanente pour être joignable à tout moment. Il ne peut donc s'exécuter sur le terminal mobile d'un utilisateur.

Les intergiciels JMS et plus généralement les MOMs actuels ne sont pas encore totalement adaptés à ces environnements. Ils se caractérisent par une couche de communication rigide et ne sont pas sensibles à leur contexte d'exécution (voir section 4.3). Généralement, ils rendent

complètement transparent aux applications le contexte dans lequel celles-ci évoluent. MobileJMS propose une approche qui répond à ces insuffisances.

5.2 Approche MobileJMS

5.2.1 Caractéristiques

MobileJMS, tout en restant compatible avec les applications écrites pour les intergiciels JMS standards, reprend une partie des idées et des concepts, que nous avons définis dans la section 4.4, relatifs à la conception des intergiciels mobiles. Nous résumons ci-après ses caractéristiques.

Adaptabilité et sensibilité au contexte L'adaptabilité d'un système mobile se traduit par sa capacité à réagir aux changements fréquents et impromptus qui peuvent se produire dans son contexte d'exécution (bande passante réseau variable, changement de localisation géographique, ressources limitées sur les terminaux mobiles, services disponibles dans le voisinage, etc.). MobileJMS est doté de mécanismes permettant le maintien de qualité de service lors du changement du contexte d'exécution ou le maintien du service même avec une certaine perte de qualité. Ces mécanismes sont essentiellement basés sur la flexibilité du module de communication, la négociation et l'utilisation dynamique des services à valeur ajoutée (section 5.4). Par exemple, dans le cas d'une dégradation de la bande passante disponible, MobileJMS peut activer un service de compression de données dans le module de communication pour maintenir la même qualité de service. Par ailleurs, MobileJMS comporte un module, appelé gestionnaire de contexte, qui maintient régulièrement toutes les informations concernant le contexte de l'utilisateur. Les décisions d'adaptation sont prises en tenant compte de ces informations.

Gestion des déconnexions Contrairement aux implémentations classiques de la spécification JMS, MobileJMS propose une solution aux problèmes de déconnexions fréquentes. Les évènements de déconnexion sont gérés par l'intergiciel et masqués aux applications JMS qui peuvent continuer leur exécution. Nous avons introduit à ce propos le service de stockage et ré-émission. Nous avons également exploité le fait qu'il n'existe pas de lien indissociable entre une connexion JMS et la connexion réseau sous-jacente. L'accès d'un client à un sujet ou une file d'attente ne dépend pas de son adresse sur le réseau. En outre, la nature asynchrone de la communication garantit qu'aucun message destiné à un client ne sera pas perdu pendant la période de sa déconnexion. La gestion de déconnexion est assurée par le mécanisme de reconnexion transparente (section 5.8).

Accès multi-réseaux La structure du module de communication permet d'associer simultanément plusieurs connexions physiques avec une seule connexion JMS (section 5.6). Une application peut interagir avec le fournisseur JMS à travers plusieurs connexions réseau (802.11b, GPRS, Ethernet, etc.) ou à travers plusieurs protocoles de transport (TCP, UDP, HTTP, etc.). Cet aspect de MobileJMS permet à un client JMS, à un moment donné, de choisir le type de

connexion qui correspond le mieux aux exigences de coût et de qualité de service que définissent les applications et les utilisateurs.

Configurabilité et reconfigurabilité MobileJMS propose aussi bien un aspect statique de configuration qu'un aspect dynamique (reconfiguration). L'aspect statique permet de décrire au démarrage de la plate-forme la structure et les services du module de communication. Il permet également d'ajuster le comportement d'un composant en particulier. L'aspect dynamique (section 5.9) permet de modifier le comportement du module de communication pendant l'exécution.

Légèreté et gestion de ressources Dans un système JMS, la majeure partie des fonctionnalités et des services (persistance, transaction, nommage, etc.) est assurée par le fournisseur. Il est ainsi possible d'obtenir une implémentation assez légère de la partie client de JMS qui est destinée à prendre en charge les applications sur les terminaux des utilisateurs. MobileJMS, tout en assurant l'essentiel des services définis par la spécification JMS, adopte une implémentation légère qui vise à améliorer la vitesse d'exécution et à minimiser l'usage des ressources de la machine. Par ailleurs, MobileJMS exploite le composant de gestion de ressources de Jonathan qui permet une gestion plus efficace de la mémoire, du codage de données et de l'ordonnancement des tâches.

Ouverture et extensibilité Compte tenu de la séparation claire entre les abstractions et implémentations qui les concrétisent, MobileJMS offre un cadre générique qui permet, sans une très profonde modification du code existant, d'introduire de nouveaux composants ainsi que de mettre à jour et de remplacer les composants existants. Ceci est particulièrement vrai pour les composants utilisés dans la communication.

Nous décrivons dans les sections suivantes ces caractéristiques à travers la description de plusieurs mécanismes liés à la communication, à l'accès multi-réseaux, à la gestion des déconnexions et à la configuration.

5.2.2 Architecture

L'architecture de MobileJMS peut être comparée, dans le domaine de la compilation, à l'articulation entre la partie frontale (*frontend*), qui est chargée d'analyser le langage en entrée, et la partie dorsale (*backend*) qui est chargée de la génération du code. Dans MobileJMS, la partie frontale est chargée d'appliquer la logique JMS en construisant les messages (requêtes) nécessaires au déroulement de l'application, la partie dorsale est chargée de transporter ces messages.

La partie frontale

La logique JMS est assurée essentiellement par OpenJMS [81] qui est une implémentation libre de la spécification JMS. Cette approche nous a permis, d'une part, d'accélérer le processus de développement, et d'autre part de reproduire intégralement les sémantiques et les services

définis par la spécification JMS. OpenJMS est destiné aux environnements fixes, mais possède des propriétés qui facilitent son extension et son optimisation. En particulier :

- Architecture modulaire et flexible à travers la séparation entre les interfaces et les composants,
- Utilisation de patrons de conception (*design patterns*), comme le *Singleton*, l'usine d'objets et le *Proxy* [29], qui facilitent l'extension et la spécialisation.
- Compatibilité avec différentes couches de transport : TCP, HTTP et Java/RMI.

Nous avons repris dans MobileJMS uniquement la partie d'OpenJMS qui ne concerne pas les aspects liés à la mobilité et au transport de messages. Cette partie inclut la construction et l'interprétation des messages JMS et les fonctionnalités assurées par le fournisseur JMS (persistance, transaction, gestion des files d'attente et des sujets, gestion des priorités, etc.).

La partie dorsale

Cette partie forme le module de communication de MobileJMS. Ce module se charge de l'échange de messages entre les clients et le fournisseur JMS. Il se base sur l'architecture du composant de communication de Jonathan. Ce choix se justifie par :

- La structure : la communication dans Jonathan est essentiellement basée sur le concept de pile de protocoles (voir section 3.5.2). Les messages sont traités successivement par chaque couche de la pile. Chaque couche n'a de lien qu'avec les couches immédiatement supérieures ou inférieures.
- La modularité et l'extensibilité : il est possible d'implémenter de nouvelles couches protocolaires ou de modifier celles qui existent déjà.
- La configurabilité : Jonathan fournit l'aspect statique de la configuration qui permet de décrire la structure de la pile de protocoles (section 3.5.3).

MobileJMS développe l'architecture de ce composant de communication en introduisant la possibilité de construire un graphe de protocoles. La différence avec une pile est qu'une couche peut être en relation directe avec plus d'une couche inférieure et plus d'une couche supérieure. Suivant son contenu et l'identité de son émetteur, un message peut être dirigé vers la couche suivante correspondante. Néanmoins, nous n'avons porté aucune modification sur les interfaces exportées par le composant de communication.

Nous avons introduit également la possibilité de modifier la structure et les services fournis par le graphe de protocoles dynamiquement. Cette possibilité permet de MobileJMS de réagir face à un état donné du contexte (adaptabilité).

5.3 Architecture du module de communication

Nous présentons dans cette section l'architecture du module de communication de MobileJMS. Cette architecture est décrite par le chemin que parcourent les messages à partir des

applications JMS jusqu'aux couches de transport réseau.

La figure 5.1 décrit le schéma détaillé des objets impliqués dans la communication d'une application basée sur MobileJMS. Nous distinguons trois types d'objets : les objets JMS, les mandataires et les objets du graphe de protocoles.

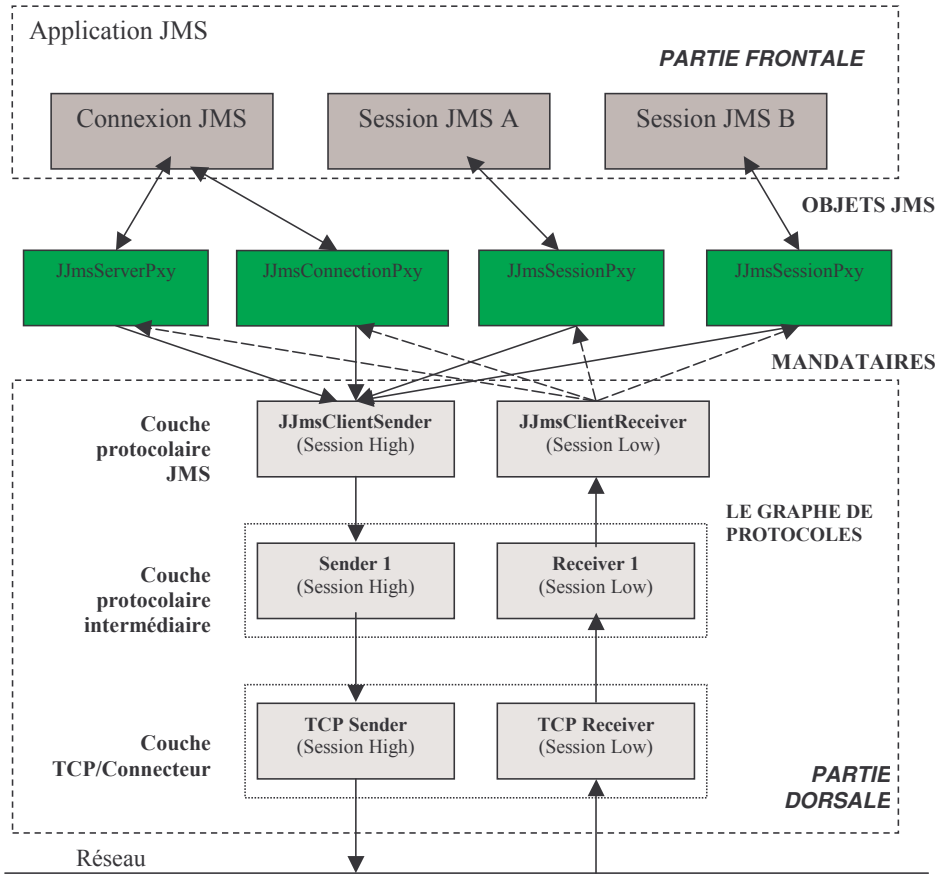


Fig. 5.1 – Exemple d'un module de communication d'un client MobileJMS

- **Les objets JMS** (composants OpenJMS) : ces objets sont instanciés par le client JMS. Nous pouvons distinguer les connexions et les sessions JMS. L'implémentation de ces objets est fournie par OpenJMS.
- **Les mandataires** (composants MobileJMS) : ces objets ont été introduits par MobileJMS afin de faire le lien entre les objets JMS et les couches de communication. Toutes les opérations des objets JMS, nécessitant l'envoi d'un message au fournisseur (création d'une connexion, création d'une session, envoi d'un message, etc.), sont encapasulées et adaptées avant d'être remis au graphe de protocoles. Les mandataires reçoivent également les messages en provenance du fournisseur avant de les remettre aux objets concernés. Les mandataires vérifient, en outre, qu'une réponse de la part du fournisseur correspond bien à la requête envoyée en utilisant un mécanisme de numérotation. Chaque mandataire est associé à un seul objet JMS.

- **Le graphe de protocoles** (interfaces Jonathan + composants MobileJMS) : Au niveau supérieur du graphe, nous trouvons les deux objets `JJmsSender` et `JJmsReceiver` qui constituent la couche protocolaire JMS. Ils implémentent respectivement les interfaces `Session_High` et `Session_Low`. Comme nous l'avons souligné dans la section 3.5.2, ces objets sont responsables respectivement de l'envoi de messages vers les couches inférieures et de l'envoi de messages vers les couches supérieures. Au niveau inférieur du graphe se trouvent les objets connecteurs qui assurent généralement un service de transport réseau (TCP, UDP, HTTP, etc.). Entre ces deux niveaux se trouvent des couches intermédiaires qui peuvent effectuer des traitements spécifiques sur les messages (compression, fragmentation, etc.). Les objets `Session_High` et `Session_Low`, d'une couche protocolaire, font partie d'un objet `Protocol` qui comporte tous les services fournis par la couche.

Le graphe de protocoles est construit par l'établissement des liens entre les objets `Session_High` et `Session_Low` des différentes couches protocolaires. Dans le cas de l'envoi d'un message vers le réseau, l'objet `Session_High` de la couche courante est associé à un seul objet `Session_High` de la couche inférieure qui doit recevoir le message. Dans le cas de la réception d'un message, l'objet `Session_Low` de la couche courante est associé à plusieurs objets `Session_Low` des couches supérieures. Suivant le chemin qui a été négocié, lors de l'établissement de la connexion, l'objet `Session_Low` courant détermine la couche supérieure qui doit recevoir le message.

Ci-dessous, quelques-unes des méthodes définies par ces objets. Le but n'est pas de décrire exhaustivement ces méthodes mais de montrer l'articulation entre les objets `Session_High`, `Session_Low` et `Protocol`.

L'objet `Session_High`

- `Send(Marshaller m)` : cette méthode envoie le message `m` à la couche inférieure. Le message `m` est encapsulé dans l'objet `Marshaller` qui fournit les méthodes pour emballer (écrire) différents types de données. L'objet `Marshaller` est crée par un objet qui implémente l'interface `MarshallerFactory` (usine d'objets). Parmi les usines qui sont implémentées dans Jonathan : `CDRMarshallerFactory` qui fournit le service d'emballage CDR de CORBA [79] et `STDMarshallerFactory` qui fournit le service d'emballage standard défini par Java/RMI.
- `LinkWith(Session_high sh)` : cette méthode établit un lien entre la `Session_High` courante et la session `sh` ; tous les messages sont envoyés vers l'objet `sh`.

L'objet `Session_Low`

- `Send(Unmarshaller m, Session_High sh)` : cette méthode envoie le message `m` vers la couche supérieure. Ce message est encapsulé dans un objet `Unmarshaller` qui permet de déballer (lire) différents types de données. Les objets `Unmarshaller` sont également créés par un objet `MarshallerFactory`. Cette méthode indique également à la couche supérieure qu'elle doit invoquer l'objet `sh` pour retourner une réponse à l'émetteur du message.

L'objet `Protocol`

- `ExportSession(Session_High sh)` : cette méthode initialise une nouvelle `Session_High` au dessus de la `sh` existante.
- `CreateProtocolGraph(ProtocolGraph[] subgraphs, Context hints)` : cette méthode, utilisée généralement par les entités serveur, ajoute le protocole courant au-dessus du graphe de protocoles `subgraphs` et retourne le nouveau graphe formé.
- `NewSessionIdentifier(SessionIdentifier next)` : cette méthode, utilisée généralement par les entités client, retourne un nouvel identificateur de session à partir la session inférieure `next`. Cet identificateur est utilisé ensuite par le client pour se connecter au serveur.

5.4 Services à valeur ajoutée

Cette section décrit les services à valeur ajoutée qui constituent une partie essentielle de l'architecture de MobileJMS. Ces services traitent certains évènements qui se produisent fréquemment dans les environnements mobiles, comme les déconnexions ou les variations de la bande passante, en améliorant les performances et le niveau d'utilisation de MobileJMS. Leur objectif est de réduire les ressources nécessaires à la transmission des messages ou de garantir un certain niveau de qualité de service.

Les services à valeur ajoutée appartiennent à la partie dorsale de MobileJMS. Ils sont implémentés en tant que couches protocolaires (voir section précédente) et peuvent donc être insérés (activés) ou supprimés (désactivés) dynamiquement du graphe de protocoles.

Il est important de noter que l'utilisation de ces services introduit généralement un surcoût par rapport à la vitesse d'exécution et à la consommation d'énergie du terminal du client. Il est donc approprié de considérer le rapport entre le bénéfice qu'ils apportent dans un contexte donné et le surcoût nécessaire (voir chapitre 6).

MobileJMS est doté actuellement de trois services à valeur ajoutée : compression, fragmentation, stockage et ré-émission. Ces services ne font pas une interprétation sémantique du contenu des messages mais les traitent simplement comme une suite d'octets. Ils peuvent être utilisés indépendamment les uns des autres et empilés dans n'importe quel ordre. Notons que l'activation d'un de ces services au niveau du client entraîne systématiquement son activation au niveau du fournisseur afin d'accomplir le traitement inverse (par exemple, compression/décompression). Une exception toutefois concerne le service de stockage et ré-émission qui ne peut être activé qu'au niveau des clients.

5.4.1 Service de stockage et ré-émission (*store and forward*)

Principe

Le service de stockage et ré-émission (S&F) est un service permettant aux applications JMS de continuer leur exécution même en cas d'une déconnexion ou d'une faible connexion avec

le fournisseur JMS. Ce service peut donc s'avérer très utile dans un contexte caractérisé par des déconnexions fréquentes. Le principe de fonctionnement est de stocker dans un cache local, lors des périodes de déconnexion, les messages que le client doit transmettre. Ces messages sont transmis vers le fournisseur dès le rétablissement de la connexion. Les évènements de déconnexion réseau sont interceptés et gérés automatiquement par ce service, et ne sont pas transmis aux applications.

Le seuil, à partir duquel la qualité d'une connexion peut être qualifiée de faible, est un paramètre que l'utilisateur peut ajuster à l'aide du composant de configuration (voir section 5.9). Dans la suite, nous assimilerons un état de faible connexion à un état de déconnexion totale.

Dans le cas où le service du S&F serait activé dans le graphe de protocoles d'un client alors que la connexion entre le client et le fournisseur est disponible, le service n'effectue aucun traitement particulier. Il se contente d'acheminer les messages reçus de la couche supérieure vers la couche inférieure. Le S&F distingue trois types de messages JMS :

- **Messages bloquants :** ces messages doivent être transmis de manière effective au fournisseur pour que la connexion ou la session JMS correspondante puisse poursuivre son traitement. Par exemple, les messages de contrôle, comme la création ou la fermeture d'une session JMS, doivent être acquittés par le fournisseur, avant que l'application ne puisse continuer son déroulement.
- **Messages non bloquants :** ces messages peuvent être stockés dans le cache local du S&F pendant un temps indéfini sans que cela n'entraîne la suspension de l'exécution de la connexion ou de la session correspondante. L'envoi de certains messages de données peut, par exemple, être différé à un moment ultérieur.
- **Messages temporisés :** ces messages doivent être transmis au fournisseur dans un certain délai. Si ce délai expire avant l'envoi du message, le service S&F envoie une exception à la connexion ou la session concernée. Ce type de message peut correspondre à une application multimédia qui doit respecter des contraintes temporelles pour la transmission de messages.

Au niveau du client, la fermeture d'une session JMS ne peut s'effectuer proprement avant que tous les messages, lui appartenant et présents dans le cache, ne soient envoyés au fournisseur. En effet, les applications supposent que ces messages ont été déjà envoyés au fournisseur. La fermeture d'une session fait partie du mécanisme de reconnexion, que nous traitons dans la section 5.8. Mais nous pouvons, dès à présent, souligner que si la connexion réseau entre le client et le fournisseur est définitivement coupée, la session JMS reçoit une exception indiquant les messages qui n'ont pas pu être envoyés.

Le type de message est spécifié par l'application dans le champ « propriété » du message JMS. Nous avons défini un ensemble de critères concernant l'ordonnancement des messages dans le cache durant les périodes de déconnexion. Ces critères fixent l'ordre d'envoi de messages après la reconnexion.

Notons que l'ordre d'envoi de messages à partir d'une même session d'un client JMS vers une même destination (file d'attente ou sujet) ne peut être différent que celui défini par cette session. Nous rappelons que la session est un contexte ordonné qui n'autorise pas des activités

concurrentes. Seul l'ordre des messages envoyés vers des destinations différentes, que ce soit à partir de la même session ou à partir de sessions différentes, peut être modifié par le service de S&F.

Le premier critère qui doit être appliqué pour l'ordonnancement est le type de message. L'ordre d'envoi est : messages bloquants, messages temporisés, puis messages non bloquants. Les messages bloquants doivent être envoyés en premier car ils nécessitent une réponse de la part du fournisseur afin que la session correspondante puisse continuer son déroulement. Les messages temporisés sont prioritaires par rapport aux messages non bloquants car ils sont soumis à des contraintes temporelles.

Les autres critères appliqués sont :
- **Niveau de priorité :** il est défini par la spécification JMS (de la priorité la plus basse 0 à la la plus haute 9) ;
- **Persistance du message :** les messages qui doivent être stockés de manière persistante par le fournisseur sont généralement plus prioritaires ;
- **Taille du message :** les messages de faible taille ont plus de chance d'être remis au fournisseur en cas de faible connexion. Ils sont donc plus prioritaires ;
- **Temps d'attente dans le cache :** les messages les plus anciens sont plus prioritaires.

L'ordre d'application de ces quatre critères peut être configuré dynamiquement par l'utilisateur (section 5.9). Si deux messages possèdent la même priorité par rapport à un critère donné, le critère suivant doit être appliqué et ainsi de suite.

Implantation

Le S&F crée une file FIFO pour chaque couple session/destination, car l'ordre des messages appartenant à ce couple ne peut être modifié. Le message transmis en premier est celui qui est le plus prioritaire parmi tous messages en tête des files FIFO. L'instanciation du service S&F se traduit par la création d'un *thread* principal qui assure la gestion du cache. Dans l'état déconnecté, ce *thread* reçoit un message de la couche supérieure, vérifie s'il est bloquant ou non, et le stocke dans la file correspondante (en créant une nouvelle file si cela est nécessaire). Si le message est non bloquant, le *thread* principal génère un « faux » acquittement qu'il envoie à la session JMS, lui permettant ainsi de reprendre son exécution. Ce « faux » acquittement remplace, en quelque sorte, celui qui est reçu habituellement du fournisseur JMS. Si le message est bloquant, aucun acquittement n'est généré et la session JMS reste bloquée jusqu'à ce que la connexion soit rétablie.

Lorsque la connexion est rétablie, le *thread* principal est interrompu en attendant que le cache soit vidé. Il faut, en effet, veiller à ce qu'aucun nouveau message reçu de la session JMS supérieure ne puisse dépasser un message du cache qui n'a pas été encore transmis. Le vidage du cache s'effectue donc à l'aide d'un *thread* secondaire. A l'issue de cette opération, le *thread* secondaire s'interrompt et réactive le *thread* principal qui reprend son activité précédente.

Le cache du S&F possède une capacité maximale configurable au delà de laquelle aucun autre

message ne peut être stocké. Dans ce cas, toutes les connexions et les sessions JMS deviennent automatiquement bloquées.

La figure 5.2 décrit un exemple d'exécution.Les messages ont le format suivant : [Session, Destination, Type (non bloquant NB, bloquant B), Priorité JMS, Persistance (oui P, non NP), Taille, Temps d'attente].

Les critères sont appliqués dans l'ordre : NB/B, priorité JMS, P/NP, taille et temps d'attente.

---------- Déconnexion ----------

Message	Propriétés du message	Etat du cache
m1 →	[S1, D1, NB, 1, P, 36, 41]	**S1/D1** (m1)
m2 →	[S2, D1, NB, 4, NP, 42, 14]	**S1/D1** (m1) **S2/D1**(m2)
m3 →	[S2, D1, NB, 1, P, 17, 56]	**S1/D1** (m1) **S2/D1**(m2,m3)
m4 →	[S1, D2, NB, 4, P, 23, 57]	**S1/D1** (m1) **S2/D1**(m2,m3) **S1/D2**(m4)
m5 →	[S2, D2, NB, 2, NP, 12, 31]	**S1/D1** (m1) **S2/D1**(m2,m3) **S1/D2**(m4) **S2/D2**(m5)
m6 →	[S2, D2, B, 3, P, 89, 11]	**S1/D1** (m1) **S2/D1**(m2,m3) **S1/D2**(m4) **S2/D2**(m5,m6)

---------- Reconnexion ----------

Envoi de m6 (message bloquant)	**S1/D1** (m1) **S2/D1**(m2,m3) **S1/D2**(m4) **S2/D2**(m5)
Envoi de m4 (priorité JMS 4 et persistant)	**S1/D1** (m1) **S2/D1**(m2,m3) **S2/D2**(m5)
Envoi de m2 (priorité JMS 4 et non persistant)	**S1/D1** (m1) **S2/D1**(m3) **S2/D2**(m5)
Envoi de m5 (priorité JMS 2)	**S1/D1** (m1) **S2/D1**(m3)
Envoi de m3 (priorité JMS 1, persistant et sa taille est 17)	**S2/D2**(m1)
Envoi de m1 (priorité JMS 1, persistant et sa taille est 36)	

Fig. 5.2 – Exemple de gestion des messages par le S&F

5.4.2 Service de compression

Le service de compression est destiné à optimiser l'utilisation de la bande passante en réduisant la taille des messages échangés entre les clients et le fournisseur JMS. La compression permet également de réduire la mémoire occupée si les messages sont stockés dans le cache du service de stockage et ré-émission.

Ce service utilise deux méthodes de compression : GZIP [19], qui propose un taux de compression unique, et ZIP qui permet d'utiliser des taux de compression, allant de 1 à 9 et correspondant respectivement à une compression faible et à une compression forte des données. Ces deux algorithmes effectuent une compression sans perte de données.

GZIP est basé sur le codage LZ77 [119] qui propose l'enregistrement de séquences de caractères dans une sorte de dictionnaires de « phrases », construit à partir des données en entrée, au fur et à mesure de la lecture du fichier à compresser. Le texte qui reste à compresser est comparé au contenu du dictionnaire. Dès qu'il rencontre une nouvelle séquence, le moteur de compression vérifie sa présence dans le dictionnaire. Si elle ne s'y trouve pas, il l'ajoute dans le dictionnaire et place en sortie un symbole marqueur qui référence son emplacement dans une table réservée à cet usage. Si la table a déjà été enregistrée, le compresseur se contente de mettre la table à jour, en y reportant l'identifiant. L'algorithme Z77 est généralement utilisé dans la compression de texte où il a montré son efficacité.

La compression en ZIP se base sur l'algorithme deflate [20] qui utilise une combinaison des algorithmes LZ77 et du codage de Huffman. Ce dernier est un système d'encodage, basé sur l'entropie, qui trouve le système optimal pour l'encodage de chaîne de caractères en tenant compte de la fréquence relative de chaque caractère.

Le service de compression de MobileJMS peut être configuré de manière à ne compresser que les messages dont la taille est supérieure à un certain seuil. Il peut être configuré également de manière à effectuer la compression ou seulement la décompression des messages reçus (comportement asymétrique).

5.4.3 Service de fragmentation

Ce service consiste à transformer des messages de grande taille en des messages ou des fragments de plus petites tailles. Ces fragments sont transmis séparément par la suite.

L'avantage de la fragmentation est que la perte de paquets ou la déconnexion n'entraîne pas la retransmission de tout le message. Seuls les paquets (fragments) perdus ou en attente sont retransmis. Du côté du destinataire, le message initial est reconstitué (défragmenté) par le service de fragmentation après la réception de tous les fragments, il est remis ensuite à la couche supérieure.

Le service de fragmentation prend également en charge les fragments dupliqués ou déséquencés. Il rajoute dans l'en-tête de chaque fragment le numéro du fragment et le nombre total de fragments.

Par ailleurs, les fragments perdus durant la transmission sont gérés de la manière suivante : dans le cas où le service de transport garantirait la fiabilité de transmission (par exemple, TCP), la perte d'un fragment ou la déconnexion génère une exception qui est interceptée et traitée par le service de fragmentation. Dans le cas où le transport ne serait pas fiable, la perte de fragments est détectée lors de la reconstitution du message initial. Si le service arrive à reconstituer les messages n+1 jusqu'à n+x alors qu'il n'a pas encore reçu tous les fragments correspondant au message n, il décide d'éliminer tous les fragments correspondant à ce message. La totalité du

message n est considérée comme perdu. Le paramètre x peut être configuré par le composant de configuration (section 5.9).

Nous verrons, dans le chapitre 6, que l'utilisation des services de compression et de fragmentation peut, suivant les cas, accélérer ou au contraire ralentir la vitesse de transmission des messages. Elle peut entraîner également une économie ou une surconsommation de l'énergie du terminal du client JMS.

5.4.4 Combinaison des services à valeur ajoutée

Les services à valeur ajoutée sont conçus de manière à s'agencer dans n'importe quel ordre dans un graphe de protocoles. Chaque combinaison de ces services peut correspondre à une configuration appropriée au type d'application de l'utilisateur et au contexte d'exécution courant. Ci-dessous, quelques-unes des configurations utiles. Chaque configuration est décrite par les couches qui la constituent en allant de la couche supérieure à la couche inférieure du graphe.

- **Configuration de base (pas de service à valeur ajoutée) :** cette configuration correspond à un contexte caractérisé par une bande passante suffisante et un très faible risque de déconnexion.
- **Compression :** cette configuration correspond à un contexte où la bande passante est insuffisante et où les ressources du terminal de l'utilisateur sont suffisantes pour effectuer la compression (notamment la puissance de calcul et la mémoire).
- **Fragmentation :** cette configuration correspond à un contexte caractérisé par une très faible bande passante, un taux de perte de paquets élevé ou un risque de déconnexion important.
- **Compression, Fragmentation :** cette configuration est semblable à la précédente mais nécessite plus de ressources machine.
- **Fragmentation, Compression :** la compression de messages de grande taille nécessite un espace mémoire qui n'est pas toujours disponible sur les terminaux légers. Dans ce cas, la fragmentation des messages, leur permet par la suite d'être compressé plus aisément.
- **S&F :** l'emploi de ce service est préconisé dans le cas où il existe un risque de déconnexion important et où l'application ne serait pas trop dépendante de la connexion réseau. Notamment, si les applications envoient régulièrement de manière asynchrone des messages qui ne nécessitent pas de réponse.
- **Compression, S&F :** cette configuration permet de réduire la taille des messages dans le cache du S&F.
- **Fragmentation, S&F :** cette configuration permet une meilleure gestion de l'envoi de fragments lors de la reconnexion et assure la continuité de l'application pendant la déconnexion.

Notons que dans ce dernier cas, le service S&F n'envoie pas un « faux » acquittement pour chaque fragment d'un même message qu'il reçoit de la part d'une session JMS. Le déblocage anticipé de cette session peut provoquer des erreurs si celle-ci envoie un autre message, alors que les fragments appartenant au message précédent n'ont pas été totalement reçus par le S&F. C'est lors de la réception du dernier fragment que le S&F envoie son « faux » acquittement.

Les fragments possèdent les même propriétés que le message à qui ils appartiennent (priorité JMS, persistance, etc.). Par conséquent, si le premier fragment est jugé comme le plus prioritaire dans le cache, tous les autres fragments le sont également.

5.5 Services de transport

Cette section décrit les services de transport de MobileJMS. Ils sont chargés d'envoyer et de recevoir les messages sur le réseau. Appelés également connecteurs, ils se situent au plus bas niveau du graphe de protocoles et encapsulent un protocole de transport. MobileJMS est actuellement doté de trois services de transport : TCP/IP, RTP (*Real-Time Protocol*) [90] et UDP. Les services TCP/IP et RTP sont fournis par le composant de communication de Jonathan. Nous avons conçu et implémenté le service UDP en adoptant les mêmes abstractions et interfaces.

TCP/IP assure un service de transport fiable en mode connecté, RTP est le protocole Internet standard pour le transport des données en temps-réel, notamment l'audio et la vidéo. RTP doit évoluer au-dessus d'un autre protocole de transport (généralement UDP).

UDP (*User Datagram Protocol*) est un protocole qui fonctionne en mode non connecté. Il n'assure ni la fiabilité ni la correction d'erreurs qui sont déléguées aux couches supérieures. Le contrôle d'erreur s'opère uniquement au niveau de l'entête du paquet (datagramme). UDP ne garantit pas, en outre, la délivrance de paquets, ce qui signifie que les pertes, les redondances et les déséquencements de paquets ne sont pas détectés et corrigés. Ce protocole est utilisé par les applications qui ne nécessitent pas l'établissement d'une connexion et qui peuvent tolérer la perte de données (par exemple, DNS et TFTP).

La différence principale dans l'utilisation des protocoles TCP et UDP réside dans l'identification du client par le serveur (le fournisseur JMS dans notre cas). Le serveur reçoit des messages adressés par plusieurs clients et doit pouvoir identifier ceux qui proviennent du même client. Dans le cas d'une connexion TCP, cette identification est simple puisque le serveur instancie un objet *Socket* pour chaque client lors de l'établissement de la connexion. Cet objet est utilisé par la suite pour l'envoi et la réception de tous les paquets entre les deux entités.

Par contre, les serveurs UDP, écrits en Java, reçoivent généralement les datagrammes en provenance des différents clients à l'aide d'un seul objet *DatagramSocket* (pas de connexion préalable). L'identification ne peut donc s'effectuer suivant l'objet utilisé par le serveur pour l'échange des datagrammes avec les clients. Dans le service UDP, que nous avons conçu, le moyen d'identification utilisé est l'adresse et le numéro de port du client qui peuvent être extraits du datagramme reçu. Ces paramètres sont associés par le service UDP à la couche supérieure correspondante.

Le service UDP constitue une extension du composant de communication de Jonathan. La mise en œuvre de ce service consiste principalement à relier les mécanismes des `Session_High` et `Session_Low` avec les objets qui encapsulent les connexions UDP. Il inclut également des propriétés de gestion de ressources comme la réutilisation des objets déjà instanciés dans le cache d'un serveur UDP.

5.6 Gestion de la multi-connexion

Cette section décrit la manière permettant à MobileJMS de communiquer simultanément et d'une manière transparente à travers plusieurs connexions réseau et plusieurs protocoles de transport.

La capacité de communiquer à travers des réseaux différents est devenue un critère de performance important des applications mobiles. En effet, de nombreuses infrastructures réseau basées sur des technologies très diverses sont apparues. Elles possèdent des caractéristiques différentes de qualité de service (QoS) telles que la fiabilité, la bande passante ou le délai de bout en bout. Ces réseaux peuvent être sélectionnés en fonction des besoins des applications mobiles. La disponibilité de plusieurs connexions permet également de mieux se prémunir contre le risque de déconnexion.

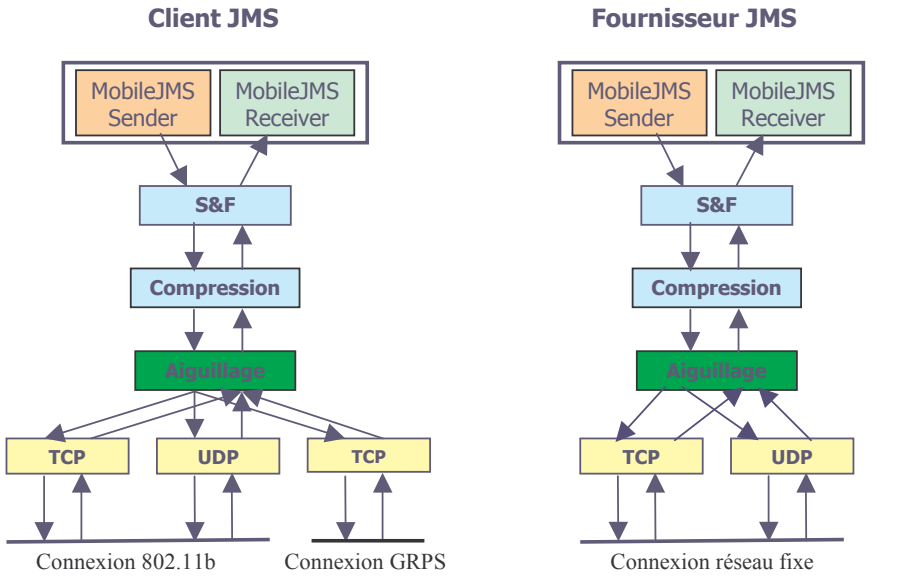
Prenons le cas d'une application de visioconférence utilisant trois connexions : une pour la transmission de la vidéo, une pour l'audio et une pour la signalisation. Typiquement, le flux vidéo est caractérisé par un débit variable avec un débit crête et un débit moyen. Son niveau de fidélité correspond à la qualité demandée par les utilisateurs (par exemple, une haute qualité pour la télévision numérique et une qualité plus faible pour les applications basées sur la localisation). Le délai de bout de bout doit être faible mais les pertes de paquets sont généralement tolérées. Le flux audio se caractérise par un débit constant avec une fiabilité presque totale et un très faible délai de bout en bout pour favoriser l'interaction entre les participants. Enfin, le flux de signalisation se caractérise généralement par un débit faible mais nécessite une connexion totalement fiable et un délai de bout en bout moins important que celui de l'audio ou de la vidéo. A chacun de ces types de flux peut correspondre un type de réseau et un protocole de transport différent. La signalisation doit être prise en charge par un protocole qui garantit une fiabilité absolue et l'acheminement des données dans l'ordre (par exemple, TCP/IP). La vidéo peut être transmise par un protocole qui ne fournit pas toutes ces garanties mais qui est moins coûteux du point de vue performance (par exemple, UDP ou RTP).

En réponse à ces besoins, le module de communication de MobileJMS prend en charge l'utilisation de plusieurs interfaces réseau ou de services de transport à la fois. Cette possibilité permet à une application JMS de communiquer simultanément à travers plusieurs connecteurs. Ces connecteurs se différencient suivant le type de réseau sous-jacent (802.11b, Bluetooth, GPRS, etc.) et le protocole de transport utilisé (TCP, UDP, HTTP, etc.). L'acheminement des messages, à travers tel ou tel connecteur, reste complètement transparent aux applications JMS. L'application ou l'utilisateur peut cependant influencer le choix du connecteur en précisant les paramètres suivants :

- Le type des messages échangés : totalement fiable/presque fiable (par exemple signalisation/multimédia).
- Les critères du choix du réseau : bande passante, délai, taux d'erreurs de transmission, coût d'accès, etc.

La figure 5.3 représente un graphe de protocole qui prend en charge plusieurs connecteurs. La communication se déroule entre un client et un fournisseur JMS. Le client utilise trois connecteurs : 2 connecteurs TCP sur les interfaces 802.11b et GRPS, et un connecteur UDP sur l'in-

terface 802.11b. Le fournisseur n'utilise, par contre, que deux connecteurs TCP et UDP sur le réseau fixe. Ces deux connecteurs représentent des entités serveur qui acceptent et établissent les connexions avec les différents clients. Nous pouvons remarquer également que tous les messages en provenance et à destination des différents connecteurs transitent par le service d'aiguillage. Il constitue l'élément clé dans le mécanisme de multi-connexion, car il fait la jonction entre les différents connecteurs et le reste du graphe de protocoles. Nous le décrivons ci-après.



(a) graphe du côté client JMS (b) graphe du côté fournisseur JMS

Fig. 5.3 – Exemple d'un graphe de protocoles multi-connecteurs

5.6.1 Service d'aiguillage

Principe

Le service d'aiguillage n'effectue aucun traitement particulier sur le contenu des messages. Son rôle est de diriger les messages. Il assure trois fonctions :

- **La sélection :** lors de l'envoi d'un message vers le réseau, il sélectionne le connecteur correspondant et lui remet le message. Cette sélection se fait par rapport aux paramètres du message (par exemple, la fiabilité) et suivant les critères du choix du réseau définis lors de l'initialisation du service.
- **Le multiplexage :** lors de la réception d'un message, il multiplexe les messages reçus des différents connecteurs et les remet aux couches supérieures correspondantes. Cette fonction est d'autant plus importante au niveau du graphe de protocoles du fournisseur JMS que celui-ci doit gérer des connexions de plusieurs clients à la fois. Le service d'aiguillage doit déterminer la couche supérieure qui correspond au client qui a envoyé le message. Nous

décrivons ce mécanisme dans la section 5.6.2.

– **L'identification :** il établit la phase d'identification. Cette phase permet au fournisseur JMS d'identifier un client quel que soit le connecteur que celui-ci utilise.

Phase d'identification

Dans un schéma à un seul connecteur, l'identification de l'entité qui a envoyé le message se fait par rapport à l'identité de l'objet `Session_High` reçu en paramètre lors de la réception du message (figure 5.4). Nous rappelons ici la méthode utilisée pour la transmission d'un message du connecteur C vers la couche d'aiguillage A :

```
session_Low_A.send(msg, session_high_c)
```

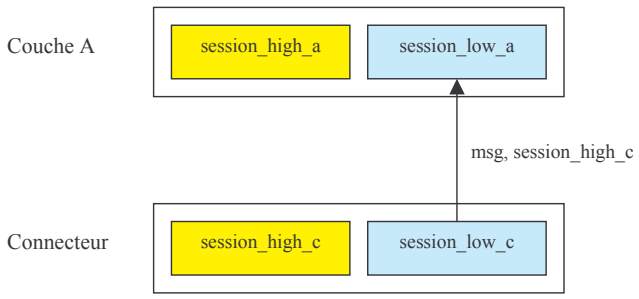


Fig. 5.4 – Identification avec un seul connecteur

Chaque objet `Session_High` d'une couche donnée désigne un destinataire unique du message. Par conséquent, la couche A établit que le message provient du client représenté par `session_high_c`. Ce schéma est récursif à travers toutes les couches protocolaires, car chaque couche n'est en rapport qu'avec les sessions des couches inférieures et supérieures. Le cheminement du message à travers le graphe de protocole se base donc sur la connaissance que possède chaque couche des sessions des couches voisines.

Dans un schéma à plusieurs connecteurs, la tâche est différente car la couche d'aiguillage A peut recevoir des messages qui appartiennent au même client à partir de plusieurs connecteurs (figure 5.5). Par exemple :

```
session_Low_A.send(msg, session_high_c1)
session_Low_A.send(msg, session_high_c2)
```

Par conséquent, l'identification à partir du connecteur de la `Session_High` n'est pas possible car il n'existe pas de moyen pour faire la correspondance entre les différentes `Session_High` désignant le même client (`session_high_c1, session_high_c2`).

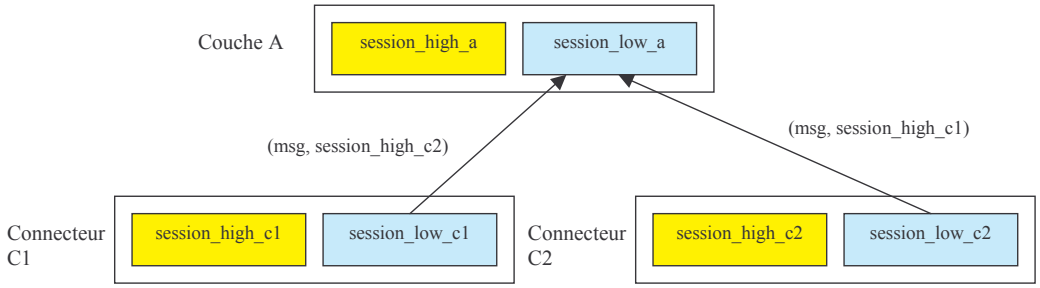


FIG. 5.5 – Identification avec deux connecteurs

Nous avons introduit dans le service d'aiguillage un mécanisme basé sur un identificateur unique attribué par le serveur (le fournisseur JMS dans notre cas). Cet identificateur doit être utilisé ensuite par le client dans tous ses échanges avec le serveur. Ce mécanisme est décrit du côté serveur et du côté client dans les figures 5.7 et 5.6. Le serveur et le client, dans cet exemple, utilisent deux connecteurs TCP et UDP.

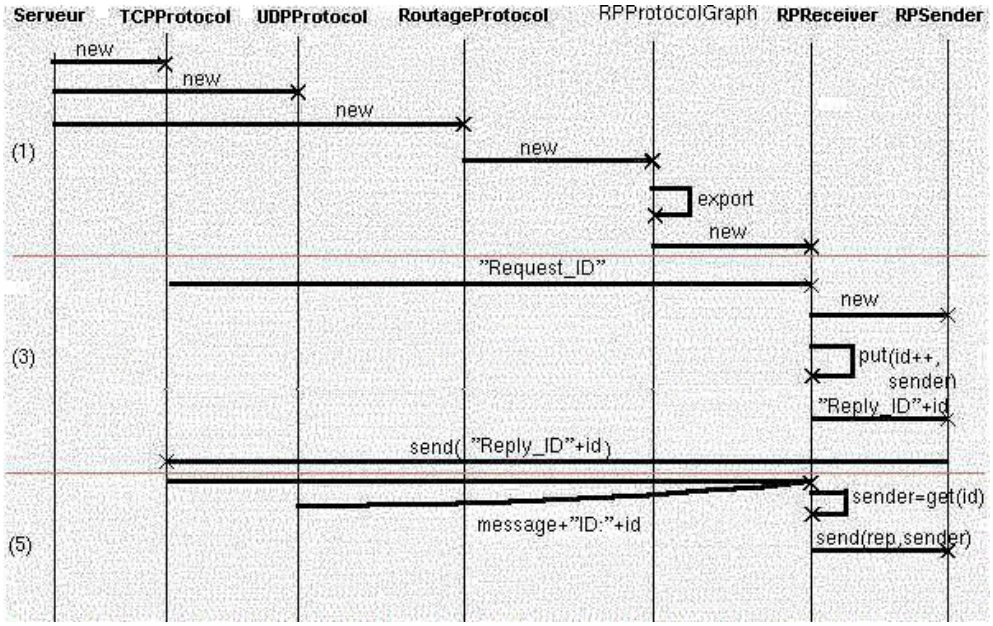


FIG. 5.6 – Mécanisme d'identification du côté serveur

L'identification se déroule selon les étapes suivantes :

1. Le serveur crée des instances des objets `TCPProtocol, UDPProtocol et RoutageProtocol` (service d'aiguillage). Il crée ensuite le graphe de protocole en faisant appel à la méthode `export`. Ce graphe est une instance de `RPProtocolGraph` (implémentation de `ProtocolGraph` dans le service d'aiguillage). Cette instance crée à son tour une instance de `RPReceiver` (implémentation de `Session_Low` dans le service d'aiguillage).

2. Le client crée également des instances des objets `TCPProtocol`, `UDPProtocol` et `RoutageProtocol` (protocole d'aiguillage). Ce dernier objet crée une instance de `RPSessionIdentifier` (implémentation de l'interface `SessionIdentifier`). Cet objet encapsule l'adresse IP et les numéros de port TCP et UDP du serveur. Le service d'aiguillage fait appel à la méthode `bind` de cet objet afin de créer uniquement une connexion TCP avec le serveur, car il n'y a pas d'établissement de connexion en ce qui concerne UDP. Nous rappelons que la méthode `bind` est utilisée pour créer la liaison (*binding*) avec un serveur (section 3.5.1). L'objet `RPSessionIdentifier` crée ensuite des instances de `RPReceiver` et `RPSender` (respectivement l'implémentation des interfaces `Session_Low` et `Session_High`). Enfin, il envoie à travers le connecteur `TCPProtocol` un message `REQUEST_ID` qui demande l'attribution d'un identificateur par le serveur. Le choix de TCP est justifié par la nécessité de transmettre ce message de manière fiable.

3. Lors de la réception du message `REQUEST_ID`, l'objet `RPReceiver` du serveur attribue un nouvel identificateur. Celui-ci est associé ensuite dans une table, réservée à cet usage, à un objet objet `RPSender` qu'il crée et qui désigne le nouveau client connecté. Il renvoie ensuite au client un message `REPLY_ID`, contenant cet identificateur, à travers l'objet `RPSender` puis `TCPIPProtocol`.
4. Le client extrait l'identificateur du message `REPLY_ID` du serveur. A chaque envoi d'un message par le client, l'objet `RPSender` met en préfixe la chaîne `"ID :"` et cet identificateur. Ce message est transmis ensuite par `TCPProtocol` ou `UDPProtocol` suivant le type du message (par exemple, fiable ou non fiable).

5. A chaque réception d'un message commençant par la chaîne `"ID :"` à partir d'un des connecteurs, l'objet `RPReceiver` du serveur extrait l'identificateur et retrouve dans sa table interne l'objet `RPSender` qui lui correspond. Cet objet est utilisé ensuite pour envoyer éventuellement une réponse à travers l'un des deux connecteurs sous-jacents.

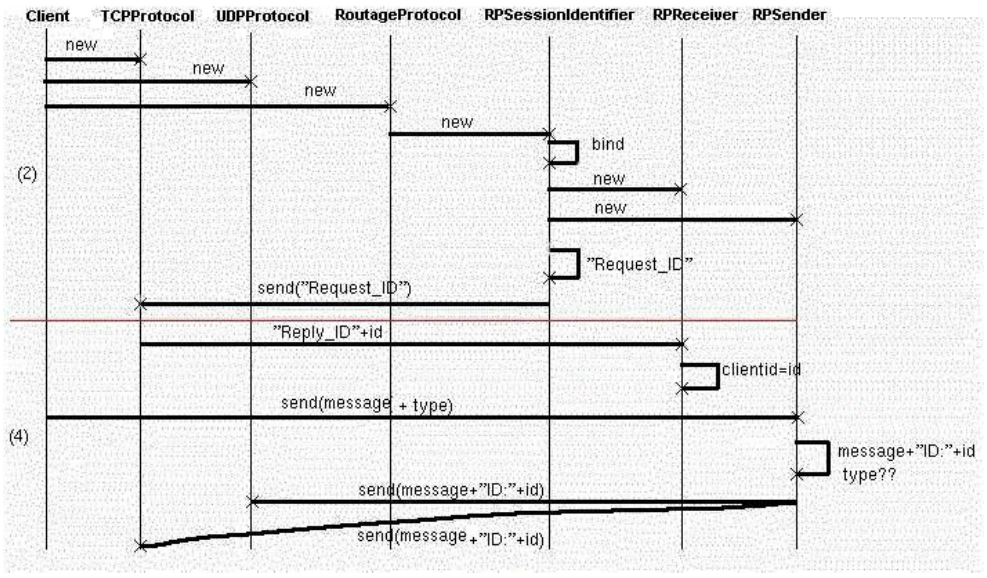


Fig. 5.7 – Mécanisme d'identification du côté client

Tous les messages échangés entre les clients et le fournisseur JMS contiennent un identificateur

unique. Cet identificateur permet également au client de pouvoir communiquer avec le fournisseur à travers un connecteur (HTTP, RTP, etc.) qui n'était pas prévu au moment de l'établissement de la connexion JMS.

5.6.2 Gestion des connexions par le fournisseur JMS

Dans MobileJMS, le fournisseur JMS doit gérer de manière concurrente la communication avec plusieurs clients. Ces clients peuvent utiliser des graphes de protocoles différents suivant leurs besoins et leur contexte d'exécution.

Dans le composant de communication de Jonathan, les messages reçus par le serveur (le fournisseur JMS dans notre cas) suivent toujours le même chemin à travers le graphe de protocoles quel que soit le client émetteur du message. Ils sont donc traités par la même séquence de couches protocolaires. Nous avons modifié ce schéma de communication afin d'instaurer des chemins multiples à travers le graphe de protocole. Par exemple, les messages du client A ont besoin d'être décompressés puis défragmentés, alors que les messages du client B n'ont besoin que d'être décompressés. Ces différents messages sont traités par un seul graphe protocolaire, seul le parcours de ce graphe est différent d'un client à l'autre.

Nous ne traitons pas, dans cette section, l'établissement du chemin dans le graphe de protocoles. Cet aspect est décrit dans la section concernant la négociation (section 5.7).

Comme nous l'avons évoqué dans la section 5.3, chaque couche protocolaire du côté du client JMS contient un objet `Session_High` (SH) et un objet `Session_Low` (SL). Puisque tous les messages sont traités de la même manière lors de l'envoi, il existe un seul chemin à travers le graphe de protocoles, entre la couche la plus haute (JMS) et la couche la plus basse (connecteurs) (à chaque objet SH d'une couche correspond un seul objet SH de la couche inférieure). Une exception concerne, toutefois, le choix du connecteur à partir de la couche d'aiguillage. Lors de la réception de messages, il existe également un seul chemin entre le connecteur et la couche la plus haute (JMS) (à chaque objet SL d'une couche correspond un objet SL de la couche supérieure).

Du côté du fournisseur JMS, le graphe de protocoles se compose des services qui sont utilisés actuellement par les clients connectés. Chaque couche contient un seul objet SL et un ou plusieurs objet SH, qui correspondent chacun à un client utilisant le service fourni par la couche. L'envoi d'un message vers le réseau n'est donc pas différent de celui du côté client, car à chaque objet SH correspond un seul objet SH de la couche inférieure. Les messages parcourent une pile d'objets SH.

En revanche, la tâche est différente lors de la réception d'un message. L'objet SL de chaque couche peut être associé à plusieurs objets SL correspondant à des couches différentes. Chaque objet SL maintient une table interne qui associe pour chaque client utilisant la couche, un objet SH de la couche inférieure et un objet SL de la couche supérieure (cette table est construite au niveau de chaque couche au moment de la négociation). Suivant la référence de l'objet SH reçu de la couche inférieure (section 5.6.1), l'objet SL de la couche courante détermine l'objet SL suivant qui doit traiter le message. Cet objet SL peut être différent pour chaque client. Les

messages parcourent donc un graphe d'objets SL.

La figure 5.8 représente un exemple de communication entre deux clients et un fournisseur JMS. Le client 1 utilise un graphe de protocoles composé d'une couche de fragmentation, d'une couche d'aiguillage et d'un connecteur TCP. Le client 2 utilise un graphe de protocoles composé d'une couche de fragmentation, d'une couche de compression, d'une couche d'aiguillage et d'un connecteur TCP. Au niveau du fournisseur, les couches de fragmentation et de compression sont également initialisées. Les couches de fragmentation et d'aiguillage contiennent chacune deux objets SH, alors que la couche de compression ne contient qu'un seul objet SH.

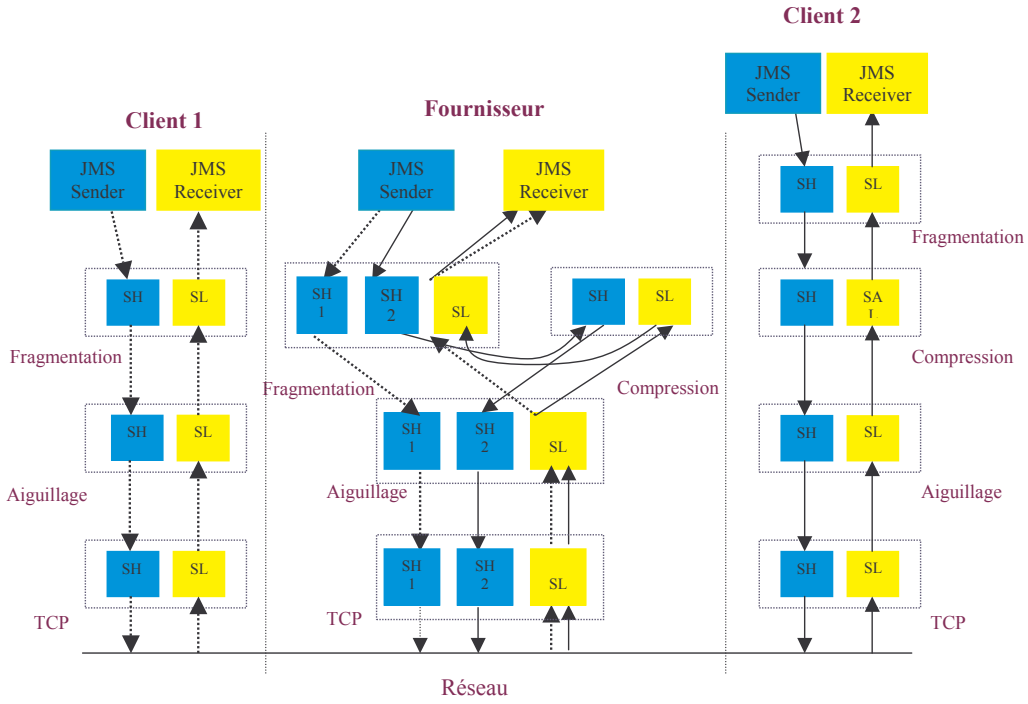


Fig. 5.8 – Communication entre deux clients et un fournisseur

Les messages échangés, entre le client 1 et le fournisseur, sont représentés par les flèches en pointillé. Les messages échangés, entre le client 2 et le fournisseur, sont représentés par les flèches pleines. Nous pouvons observer qu'une seule couche de compression au niveau du fournisseur traite les messages reçus des clients 1 et 2. Alors que la couche de fragmentation ne traite que les messages reçus du client 2. Nous avons préféré cette solution à celle qui consiste à instancier au niveau du fournisseur un graphe de protocole différent (des couches différentes) pour chaque client. Ainsi, si n clients utilisent la compression, le fournisseur doit instancier n objets SH de compression et n objets SL de décompression. Cette solution est trop lourde en terme de consommation de mémoire et introduit une latence importance lors de l'initialisation des connexions.

5.7 Mécanisme de négociation dynamique

Cette section décrit le mécanisme de négociation dynamique de MobileJMS. Ce mécanisme permet de sélectionner et de configurer de manière dynamique les services à valeur ajoutée utilisés par le client et le fournisseur lors de leurs échanges.

Principe

Le client ou le fournisseur JMS doivent pouvoir interpréter correctement les messages reçus en les traitant par les mêmes services que ceux utilisés lors de l'envoi. Par exemple, un message compressé doit être décompressé lors de la réception.

Une première solution consiste à spécifier dans l'en-tête du message les différents traitements qu'il a subis. Cependant, cette solution oblige le récepteur du message à effectuer des traitements qu'il ne peut pas ou ne souhaite pas faire. Par exemple, si le message a été compressé selon un format particulier, il est possible que le récepteur ne possède pas le service de décompression approprié. Le récepteur peut également ne pas détenir à ce moment les ressources nécessaires à la décompression. En outre, le délai nécessaire à l'instanciation d'un nouveau service peut réduire sensiblement le temps de réponse.

La solution, que nous avons retenue dans MobileJMS, consiste à établir un mécanisme de négociation entre le client et le fournisseur. Ce mécanisme sert à déterminer le graphe de protocoles (GP) qui doit être utilisé de chaque côté. Il est effectué lors de l'initialisation de la connexion JMS. Une renégociation du graphe peut être également effectuée pendant la durée de vie de la connexion JMS.

Négociation initiale

Lors de l'établissement de la connexion, la négociation se déroule de la manière suivante (figure 5.9) : le client demande l'établissement d'une connexion JMS après avoir obtenu un identificateur. Cette demande est envoyée directement sur le connecteur (section 5.6.1). Elle contient la représentation symbolique du graphe de protocoles (GP) que le client veut instancier. Le client indique également les services du GP qui dépendent d'autres services. Par exemple, le client peut indiquer que l'utilisation du service compression dépend de l'utilisation du service de fragmentation.

A la réception de cette demande, le fournisseur décide de l'accepter ou non. Trois cas sont possibles :

- Le fournisseur accepte ce GP et instancie un graphe réciproque, il envoie ensuite un message d'acceptation.
- Le fournisseur rejette complètement le GP et envoie un message de rejet.
- Le fournisseur rejette certains services dans le GP et accepte d'autres. Il vérifie s'il existe des services acceptés qui dépendent de l'un des services rejetés. Dans ce cas, il décide de

rejeter également ces services. S'il existe des services acceptés qui peuvent être utilisés indépendamment, le fournisseur instancie un GP contenant uniquement ces services. Il envoie ensuite un message contenant les services rejetés. Dans le cas contraire, il envoie un message de rejet du GP.

Suivant le contenu de la réponse, le client instancie la totalité ou seulement une partie de GP. A ce moment, le client peut envoyer les messages en utilisant le GP, car le fournisseur a déjà instancié un graphe similaire.

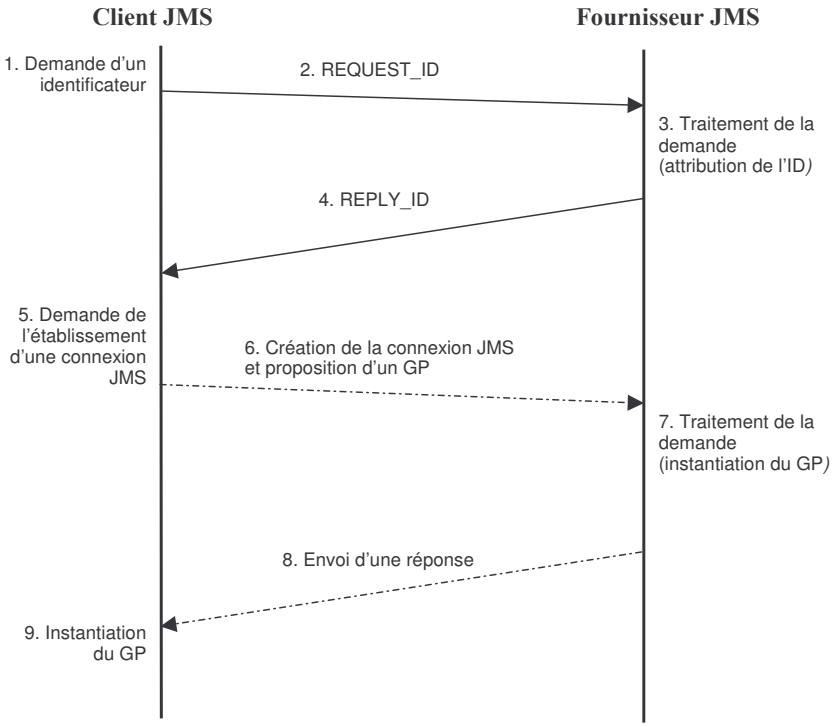


Fig. 5.9 – Déroulement de la négociation lors de l'établissement de la connexion

Renégociation dynamique

Pendant la durée de vie de la connexion JMS, une nouvelle phase de négociation peut se dérouler. Elle est déclenchée soit à l'initiative du client soit à celle du fournisseur. Le client ou le fournisseur décident généralement de renégocier suite à un changement dans le contexte d'exécution. Par exemple, un client JMS négocie l'ajout d'un service de compression dans le graphe, dans le cas où la bande passante diminuerait sensiblement. Le fournisseur peut demander, à son tour, à un client de ne plus compresser les messages car il est surchargé par les nombreux clients utiliant ce service.

La nouvelle phase de négociation se déclenche suite à une modification dans la configuration du graphe de protocole que maintient MobileJMS. Sur l'initiative du client, la renégociation se déroule de la manière suivante (figure 5.10) :

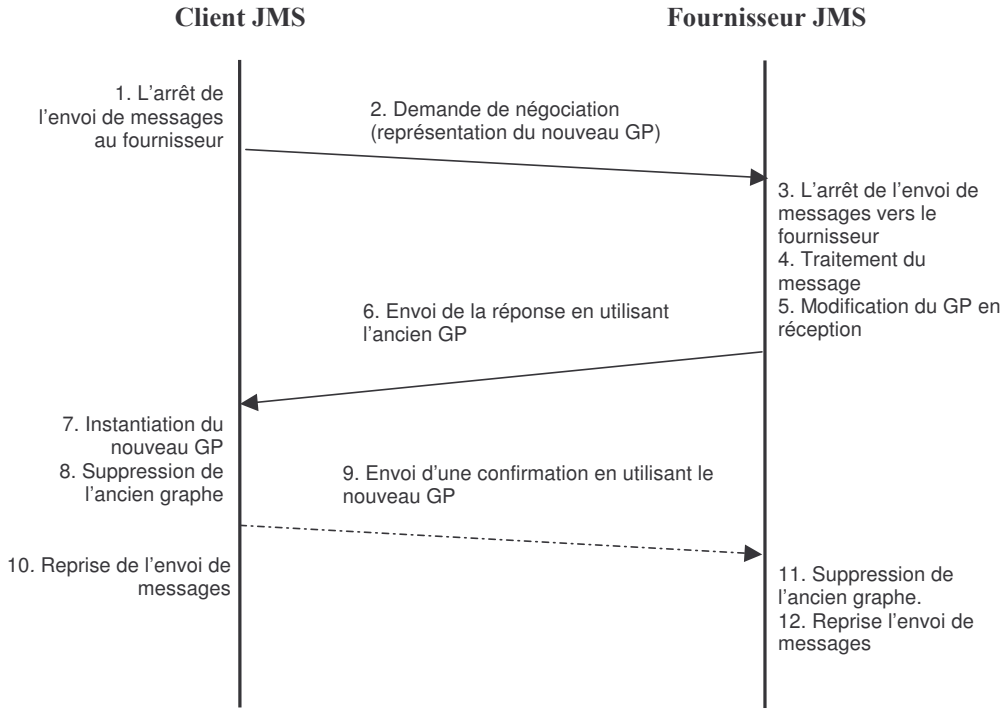


Fig. 5.10 – Déroulement de la renégociation sur l'initiative du client

- Le client arrête l'envoi de messages et envoie une demande de négociation au fournisseur contenant une représentation symbolique du nouveau graphe de protocole qu'il souhaite instancier. Si le service de S&F est instancié à ce moment dans le graphe de protocoles du client, la demande de négociation arrête également l'envoi des messages du cache car elle possède la plus haute priorité.
- A la réception de cette demande. Le fournisseur arrête également l'envoi de messages vers le client. S'il décide d'accepter cette proposition, il modifie le GP en réception pour pouvoir interpréter correctement les futurs messages envoyés par le client. S'il décide de rejeter la demande, il envoie une réponse au client et reprend aussitôt l'envoi de messages. La réponse est envoyée au client en utilisant l'ancien GP, car le client n'a pas encore instancié le nouveau GP.
- Si le client reçoit une réponse positive, il instancie le nouveau GP et supprime l'ancien. Il envoie ensuite une confirmation au fournisseur puis reprend l'envoi de messages. Cette confirmation est envoyée en utilisant le nouveau GP, elle est nécessaire au fournisseur pour mettre complètement à jour son graphe de protocoles (par exemple, les services qui ne sont plus utilisés par aucun client sont supprimés du graphe) et reprendre ensuite l'envoi de messages.
- Si le client reçoit une réponse négative, il reprend à son tour l'envoi de messages.

Sur l'initiative du fournisseur, la renégociation se déroule globalement de la même manière.

Enfin, il est à noter que l'interruption de la procédure de négociation en raison d'une déconnexion est traitée dans le cadre du mécanisme de reconnexion transparente (section suivante).

5.8 Mécanisme de reconnexion transparente

Comme nous l'avons souligné précédemment, les environnements mobiles sont souvent caractérisés par des déconnexions fréquentes et impromptues. Ces déconnexions sont considérées généralement comme des erreurs fatales par les intergiciels JMS classiques, ce qui implique la fermeture des connexions, des sessions et des autres objets JMS. L'application peut s'arrêter brusquement, si elle n'a pas prévu un mécanisme qui gère cet événement. En outre, les messages non encore transmis sont perdus, sauf s'il existe un service de stockage et ré-émission persistant. L'utilisateur est généralement obligé de redémarrer l'application lorsque la connexion redevient disponible.

Pour faire face à ce genre de défaillances, nous avons introduit dans MobileJMS un mécanisme qui permet de traiter les déconnexions, qui se produisent entre un client et un fournisseur JMS, au niveau du module de communication. Les déconnexions sont complètement masquées aux couches JMS et donc aux applications. L'évènement de déconnexion, qui est généré par les connecteurs, est intercepté et traité au niveau du graphe de protocoles. Cet événement parcourt le graphe de bas en haut.

L'interception peut s'opérer à deux niveaux :

- Au niveau du service de stockage et ré-émission (S&F) : l'application peut continuer à émettre des messages qui sont stockés dans le cache du S&F.
- Au niveau de la couche JMS du graphe : le traitement peut se faire de deux manières :
  1. Le S&F peut être inséré dans le graphe de protocoles, ce qui nous renvoie au premier cas.
  2. Le S&F ne peut pas être inséré, ce qui entraîne la suspension des connexions et sessions JMS.

  Dans les deux cas, tous les objets JMS restent instanciés au niveau du client et du fournisseur.

Lors d'une reconnexion éventuelle avec le fournisseur, le client poursuit normalement les traitements qu'il effectuait au moment de la déconnexion. Les messages stockés dans le cache du S&F sont transmis au fournisseur. Notons que le fournisseur attribue automatiquement, lors de l'initialisation de toute connexion JMS, un identificateur unique et global défini par la spécification JMS. Le client utilise ce paramètre, lors de la procédure de reconnexion, pour s'identifier auprès du fournisseur. Par conséquent, le changement de l'adresse réseau du client n'influe pas sur cette reprise. Les objets JMS du fournisseur qui communiquent avec le client sont réassociés à cette nouvelle connexion réseau.

La reconnexion d'un client utilisant le S&F se déroule de la manière suivante (figure 5.11) :

- L'événement de déconnexion est intercepté par le S&F. Tous les connecteurs réseau utilisés

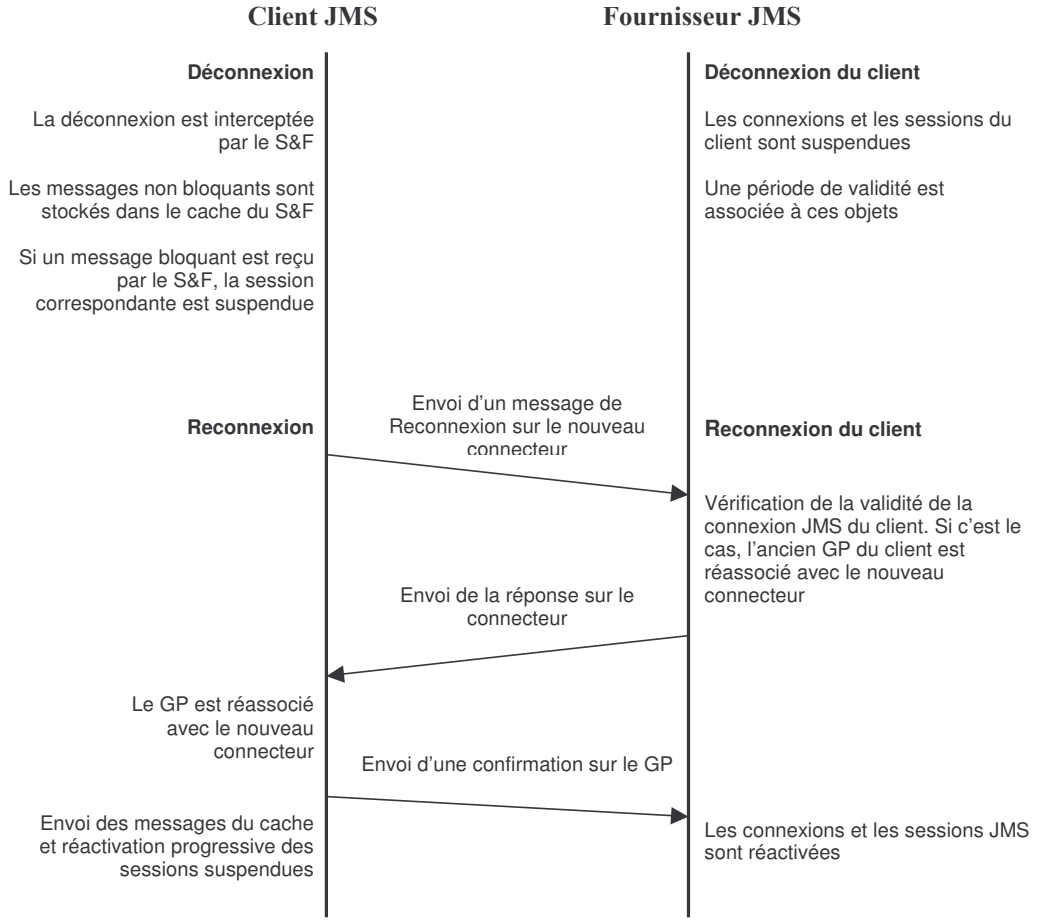


Fig. 5.11 – Exemple de reconnexion d'un client utilisant le S&F

par le client sont fermés, mais le reste du graphe de protocoles (GP) reste instancié. Le fournisseur, qui détecte également la déconnexion, suspend tous les objets JMS (sessions, connexions, etc.) associés au client. L'envoi des messages du cache du S&F est suspendu par le client. Les connexions et les sessions JMS qui envoient des messages bloquants sont automatiquement suspendues. Un délai de validité est fixé par le fournisseur : si le client ne se reconnecte pas avant son expiration, ces objets JMS sont définitivement fermés.

– Si le client est de nouveau connecté avec le fournisseur, il envoie un message de reconnexion incluant l'identificateur (ID) JMS attribué par le fournisseur lors de la première connexion. Ce message est délivré directement en utilisant le nouveau connecteur réseau.

– A la réception du message de reconnexion, le fournisseur vérifie s'il existe un client déconnecté possédant cet ID et vérifie que les objets JMS associés à ce client restent valides. Si c'est le cas, il relie le chemin du client dans son GP avec le nouveau connecteur de celui-ci. Dans les deux cas, il envoie ensuite une réponse en utilisant seulement ce connecteur.

– Si le client reçoit une réponse positive, il réassocie son GP avec le nouveau connecteur et envoie une confirmation au fournisseur afin qu'il réactive les objets JMS suspendus. Les

messages contenus dans le cache du S&F sont envoyés et les sessions suspendues sont réactivées après l'envoi des messages bloquants. Si le client reçoit une réponse négative, les connexions et sessions JMS sont définitivement fermées, ce qui entraîne l'arrêt de l'application.

La reconnexion dans le cas de l'absence du S&F suit à peu près le même déroulement, sauf que les connexions et les sessions JMS sont systématiquement suspendues au niveau du client.

Il est important de noter que les messages de négociation sont traités comme les autres messages en cas de déconnexion. Il se peut donc que la procédure de négociation soit interrompue par une déconnexion mais poursuit néanmoins son déroulement après la reconnexion. Ce cas de figure est vérifié par le modèle formel que nous décrivons dans le chapitre 7.

5.9 Configuration/reconfiguration

Nous avons présenté dans la section 3.5.3 le composant de configuration de Jonathan. Cette section décrit l'usage que fait MobileJMS de ce composant, ainsi que les extensions qui ont été introduites en ce qui concerne la reconfiguration.

La configuration

Le composant de configuration de Jonathan permet une gestion fine de la création et de l'assemblage des composants. Dans MobileJMS, la configuration est notamment utilisée pour :

- La description de la composition du graphe de protocole.
- La configuration des services à valeur ajoutée :
  - compression : spécification de l'algorithme et du taux de compression ainsi que de la taille minimum ;
  - fragmentation : spécification de la taille du fragment ;
  - S&F : spécification de l'ordre des priorités et de la taille maximale du cache.
- La configuration des ressources qu'utilisent les services : gestionnaires de connexions, usines de *marshallers*, usines de contextes, les ordonnanceurs (*schedulers*).

Voici ci-dessous un extrait d'un fichier de configuration XML :

```
<ELEM NAME="protocols_size">
  <PROPERTY VALUE="3" TYPE="int"/>
</ELEM>

<ELEM NAME="protocols">
  <CONFIGURATION>
    <ELEM NAME="protocol1">
       <ALIAS NAME="/compression"/>
    </ELEM>
    <ELEM NAME="protocol0">
```

```
      <ALIAS NAME="/fragmentation"/>
    </ELEM>
    <ELEM NAME="protocol2">
      <ALIAS NAME="/storeforward"/>
    </ELEM>
  </CONFIGURATION>
</ELEM>

...

<ELEM NAME="compression">
  <ASSEMBLAGE ALTERNATIVE="0">
    <MY_FACTORY>
      <IMPLICIT_FACTORY>
         <ALTERNATIVE ID="0"
            CLASS="org.carism.jonathan.libs.protocols.compression.CompressionProtocol"
            NAME="protocol3">
         <ARGUMENT NAME="marshaller factory"
            TYPE="org.objectweb.jonathan.apis.presentation.MarshallerFactory"/>
         <ARGUMENT NAME="compression algorithm" TYPE="java.lang.String"/>
         <ARGUMENT NAME="level" TYPE="int"/> </ALTERNATIVE>
      </IMPLICIT_FACTORY>
    </MY_FACTORY>
    <MY_CONFIGURATION>
      <CONFIGURATION>
        <ELEM NAME="marshaller factory">
          <ALIAS NAME="/resources/marshallerfactory"/>
        </ELEM>
        <ELEM NAME="compression algorithm">
          <PROPERTY VALUE="zip" TYPE="String"/>
        </ELEM>
        <ELEM NAME="level">
          <PROPERTY VALUE="1" TYPE="int"/>
        </ELEM>
        <ELEM NAME="minimum size">
          <PROPERTY VALUE="500" TYPE="int"/>
        </ELEM>
        <ELEM NAME="way">
          <PROPERTY VALUE="2" TYPE="int"/>
        </ELEM>
      </CONFIGURATION>
    </MY_CONFIGURATION>
  </ASSEMBLAGE>
</ELEM>
```

Cet extrait du fichier de configuration définit le graphe de protocoles à utiliser. Ce dernier est composé des couches suivantes (de haut en bas) : S&F, compression et fragmentation. Les couches d'aiguillage et de TCP sont ajoutées automatiquement au graphe.

La couche de compression est configurée de manière à utiliser l'algorithme ZIP avec le taux de compression 1. Le service fonctionne dans les deux sens (compression et décompression) et la taille minimale pour pouvoir compresser un message est de 500 octets. Le fichier spécifie également le nom de la classe qui implémente le service de compression ainsi que les différentes ressources qu'elle utilise.

La reconfiguration dans MobileJMS

Le composant de configuration de Jonathan permet de configurer la plate-forme uniquement au moment de l'initialisation. Le contenu du fichier de configuration XML est analysé afin d'extraire les paramètres d'initialisation des différentes classes et de faire les assemblages nécessaires.

Nous avons modifié ce comportement afin de prendre en compte les modifications qui interviennent dans le fichier au moment de l'exécution. Ainsi, nous avons introduit dans MobileJMS un gestionnaire de configuration (*ConfigurationManager*) qui maintient et met à jour les paramètres de configuration. Il utilise un *thread* qui est chargé d'obtenir régulièrement la configuration actuelle. Si les paramètres de configuration sont modifiés, le *ConfigurationManager* déclenche une procédure de négociation dynamique (section 5.7) avec le fournisseur.

Afin d'éviter les problèmes d'écritures/lectures concurrentes, le fichier ne peut pas être lu ou modifié manuellement. Seul un module, nommé serveur de configuration, est responsable de l'écriture et de la lecture du fichier. La configuration de MobileJMS peut être modifiée par une tierce partie : l'utilisateur ou le gestionnaire d'adaptation (voir chapitre 6), en communiquant les nouveaux paramètres au serveur de configuration. Le *ConfigurationManager* reçoit alors une notification du serveur de configuration contenant ces nouveaux paramètres. Le serveur synchronise les requêtes qu'il reçoit des différentes parties.

5.10 Un scénario d'utilisation de MobileJMS

Nous présentons dans cette section un scénario d'usage de MobileJMS dans un contexte nomade (figure 5.12).

Pierre est muni d'un PDA doté d'une interface Bluetooth et d'une interface 802.11b. Pendant le trajet de train qui le conduit à Paris, il allume son PDA et lance une application, basée sur MobileJMS, qui sert à rechercher et à réserver les chambres d'hôtel disponibles ainsi qu'un plan pour s'y rendre. Le PDA se connecte au réseau à travers l'interface Bluetooth qui le relie à un téléphone mobile offrant un accès GPRS. Après l'établissement de la connexion JMS avec le fournisseur JMS, l'application envoie une requête à un fournisseur de services touristiques pour rechercher toutes les chambres d'hôtels disponibles près de la gare. Cette requête est stockée dans la file d'attente FS du fournisseur. Notons que le graphe de protocoles du client est composé d'un service de S&F, d'un service de compression, d'un service d'aiguillage et d'un connecteur TCP/Bluetooth (figure 5.13 avant).

Compte tenu du temps que le traitement de cette requête risque de prendre et de la bande passante moyenne de ce réseau, Pierre décide de se déconnecter afin d'économiser les batteries et de réduire le temps de connexion. MobileJMS ferme le connecteur mais l'application continue de fonctionner en mode déconnecté TCP/Bluetooth (figure 5.13 avant). Pendant ce temps, le fournisseur traîte la requête et envoie sa réponse en la stockant dans la file d'attente CLT du client au niveau du fournisseur.

A son arrivé à la gare, MobileJMS constate qu'un réseau 802.11b est disponible. Après l'acti-

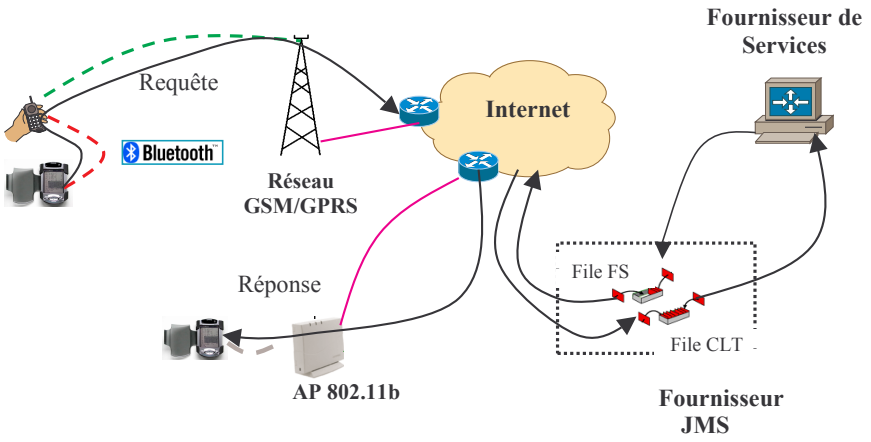


Fig. 5.12 – Exemple de gestion de la mobilité

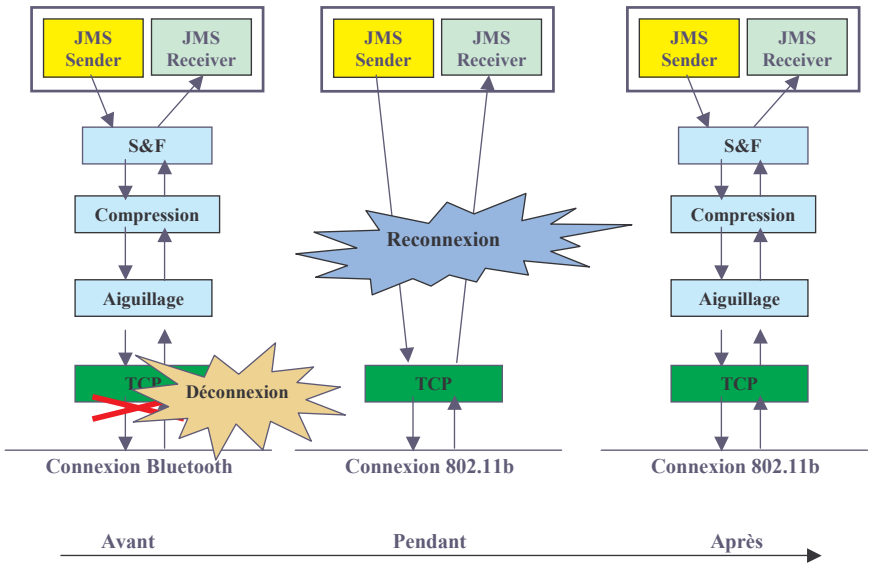


Fig. 5.13 – Graphe de protocoles lors de la déconnexion/reconnexion

vation d'une connexion 802.11b, il instancie un nouveau connecteur en utilisant cette connexion. Il déclenche aussitôt la procédure de reconnexion avec le fournisseur JMS (figure 5.13 pendant). Ce dernier l'accepte et lui envoie la réponse du fournisseur de services touristiques. Le graphe de protocole du client est réassocié avec ce nouveau connecteur (figure 5.13 après). La nouvelle bande passante disponible permet à Pierre de consulter et de visualiser les plans des hôtels trouvés avec une bonne qualité.

5.11 Synthèse

Nous avons présenté dans ce chapitre les différents aspects relatifs à l'architecture et à l'implémentation de MobileJMS. La communication entre les différentes entités est basée sur les abstractions fournies par la plate-forme Jonathan. MobileJMS met au point la notion de graphe de protocoles pour permettre un traitement différencié pour chaque client suivant son contexte. La structure de ce graphe peut être modifiée pendant l'exécution. Cette forme de communication permet à MobileJMS de prendre en charge quelques-unes des problématiques importantes des environnements nomades :

- **Gestion de la variabilité des conditions de transmission :** MobileJMS réagit à cette variabilité (bande passante, perte de paquet) en utilisant de manière dynamique les services à valeur ajoutée (compression, fragmentation). Toute modification qui se produit sur la structure du module de communication déclenche une phase de négociation entre le client et le fournisseur. Ces modifications, qui peuvent être automatiques ou à l'initiative de l'utilisateur, sont notifiées à MobileJMS à l'aide du composant de configuration.

- **Gestion des déconnexions fréquentes :** les déconnexions brèves sont occultées aux applications qui continuent de suivre un comportement normal. Le mécanisme de reconnexion dynamique permet de gérer les phases de déconnexion/reconnexion sans affecter les connexions et les sessions JMS associées. Le service de S&F stocke les messages temporairement dans un cache local du client avant leur transmission effective lors de la reconnexion. Ce service introduit des critères pour la gestion de ces messages. Par exemple, si l'environnement se caractérise momentanément par des connexions brèves et de faible qualité, le S&F favorise l'échange de messages, et par conséquent les applications, qui sont les plus prioritaires pour l'utilisateur.

- **Gestion de l'accès multi-réseaux :** le module de communication de MobileJMS permet d'associer à une application JMS de multiples connexions réseau (802.11b, GPRS, Ethernet, etc.) et de multiples protocoles de transport (TCP, UDP, HTTP, etc.). Le graphe de protocoles contient une couche spécifique (aiguillage) qui permet de rediriger les messages, suivant leurs propriétés (fiable, multimédia, etc.), vers le protocole de transport et l'interface réseau la plus adaptée. L'accès multi-réseaux permet également de réduire les risques d'arrêt des applications en basculant automatiquement le transport des messages sur un deuxième réseau si une déconnexion se produit dans le premier.

Les mécanismes de négociation et de reconnexion font l'objet d'une validation formelle dans le chapitre 7.

Enfin, les capacités de MobileJMS, de s'adapter face à certains évènements qui se produisent dans le contexte d'exécution, représentent un préalable pour mettre en œuvre un modèle d'adaptation plus globale. En effet, nous abordons dans le chapitre suivant la politique qui permet de traduire un certain état du contexte d'exécution en un comportement (configuration) donné de MobileJMS.

Chapitre 6

Le modèle d'adaptation de MobileJMS

Nous avons constaté au cours du chapitre précédent que l'intergiciel MobileJMS est doté de mécanismes qui lui permettent de s'adapter à certaines conditions du contexte d'exécution. Ces mécanismes représentent des briques de base, mais il nous reste à définir la politique qui enclenche et met en œuvre les différentes décisions relatives à l'adaptation.

L'objectif de ce chapitre est de proposer un modèle d'adaptation qui met en œuvre une coopération entre l'utilisateur, les applications et MobileJMS. Nous avons pu établir, à travers l'état de l'art, que l'adaptabilité d'un système mobile ne peut être gérée efficacement qu'à travers une approche transversale qui met à contribution toutes les couches du système. Les décisions d'adaptation peuvent être de natures différentes et doivent se prendre qu'à un niveau qui détient une vue globale. Il existe en effet des décisions qui relèvent de l'intergiciel, comme la compression des messages, et des décisions qui relèvent du niveau applicatif, comme la modification du type des données échangées. Nous avons conçu, dans ce but, un module, appelé gestionnaire d'adaptation, qui centralise toutes les informations et la logique intervenant dans l'adaptation de la plate-forme. Nous présentons d'abord l'approche du modèle d'adaptation et les concepts qui lui sont liés, comme les politiques d'adaptation. Nous traitons ensuite de la mise en œuvre de ce modèle à travers le gestionnaire d'adaptation.

Un autre aspect abordé par le chapitre est la définition et la mise en œuvre de deux critères de performance qui font partie du modèle d'adaptation et qui influent sur la configuration de MobileJMS. Le premier critère est la vitesse de transmission des messages échangés entre les couches JMS du client et du fournisseur. Le deuxième critère est la consommation d'énergie nécessaire à la transmission des messages par le client. Nous exposons, notamment, dans quelles conditions l'utilisation des services de compression et de fragmentation peut améliorer cette vitesse ou réduire la consommation de l'énergie sur le terminal de l'utilisateur. Nous définissons à ce propos un modèle de performance, simple à mettre en œuvre, permettant d'évaluer ces deux critères de performance.

Enfin, dans la dernière partie, nous exposons la projection des modèles d'adaptation et de performance en terme de configuration et des services fournis par MobileJMS.

6.1 Approche

Une application MobileJMS est un ensemble de fonctionnalités fournies par un ou plusieurs services. Un service peut être défini comme une *« partie distincte des fonctionnalités qui sont fournies par une entité à travers des interfaces »* [47]. Les *interfaces* sont un ensemble d'opérations qui caractérisent le comportement d'une entité. Ces *opérations* sont la spécification d'une transformation ou d'une requête que l'entité peut être amenée à exécuter. Chaque opération possède un nom et une liste de paramètres. Par ailleurs, un service est aussi un nom générique d'une fonctionnalité qui peut être implémentée par plusieurs fournisseurs. Par exemple, le service, qui permet de se guider dans une ville ou de trouver l'hôtel le plus proche, peut être implémenté par plusieurs agences touristiques ou services d'annuaires.

Les services se caractérisent également par le mode avec lequel ils interagissent avec les serveurs distants : interaction synchrone/asynchrone, code mobile, technologies agent, etc. Dans notre cas, les services sont implémentés sous la forme de clients JMS effectuant des opérations à travers des requêtes asynchrones adressées à un serveur. Ces services peuvent également être utilisés par une ou plusieurs applications à la fois.

Nous avons conçu un modèle d'adaptation qui met en œuvre une coopération entre les utilisateurs, les services et l'intergiciel MobileJMS. De leur part, les services spécifient un ensemble de besoins et de contraintes qui leur assurent un certain niveau de qualité de service. De sa part, MobileJMS veille à ce que ces besoins et contraintes soient respectés et notifie le service dans le cas où une violation des contraintes se produirait à un moment donné. Par ailleurs, la configuration qu'adopte MobileJMS pour répondre aux besoins des services et aux contraintes du contexte doit également tenir compte du type d'usage que souhaite l'utilisateur. Celui-ci peut souhaiter obtenir le maximum de qualité de service ou au contraire préférer l'économie de l'énergie du terminal. Chacun de ces critères constitue un mode d'exécution du service.

Par exemple, considérons que la réception de la vidéo en couleur constitue la meilleure qualité de service. Celle-ci nécessite parfois l'utilisation de la compression qui peut se révéler parfois gourmande en énergie. Par contre, la réception de la vidéo en noir et blanc constitue une qualité plus faible. Elle nécessite moins de bande passante et n'entraîne pas l'utilisation de la compression, ce qui peut permettre un gain en énergie. L'utilisateur doit donc indiquer le critère qu'il souhaite privilégier entre la qualité de service et l'énergie.

Quatre classes de paramètres interviennent dans le modèle :

- **Les besoins du service :** cette classe définit les paramètres de qualité de service (bande passante, latence maximum tolérée, etc.) et les paramètres qui sont liés à la logique fonctionnelle de l'application (adresse du serveur distant, type des messages échangés, etc.)
- **Le contexte d'exécution :** cette classe représente les paramètres du contexte d'exécution du service. Ils peuvent être de type externe à la machine (localisation géographique, type de connexion réseau, bande passante disponible, taux de perte de paquets, etc.) ou de type interne (niveau de batterie, occupation de la mémoire, utilisation du CPU, etc.).
- **Le mode d'exécution :** cette classe représente le mode d'exécution du service. Celui-ci fait référence au critère de performance que souhaite privilégier l'utilisateur pour l'exécution

du service et de l'intergiciel. Parmi ces modes :
- Maximiser la vitesse de transmission des messages (qualité de service) entre le client et le fournisseur. Cette vitesse dépend de la bande passante réseau mais aussi du temps de traitement sur le terminal qui héberge le client, comme le temps nécessaire à la compression des messages ;
- Minimiser la consommation de l'énergie du terminal.

– **La configuration de l'intergiciel :** cette classe représente la configuration qu'adopte l'intergiciel à un moment donné pour répondre à la fois aux besoins des services et au mode d'exécution, et pour faire face aux conditions du contexte d'exécution. La configuration se traduit, par exemple, par utilisation de la compression des messages, de la fragmentation ou du service de stockage et ré-émission.

La figure 6.1 représente la fonction principale du modèle d'adaptation qui consiste à traduire les besoins du service, le mode d'exécution et l'état du contexte d'exécution, en une certaine configuration de MobileJMS.

Il est à noter que ce modèle d'adaptation concerne uniquement les entités qui s'exécutent sur le terminal mobile de l'utilisateur.

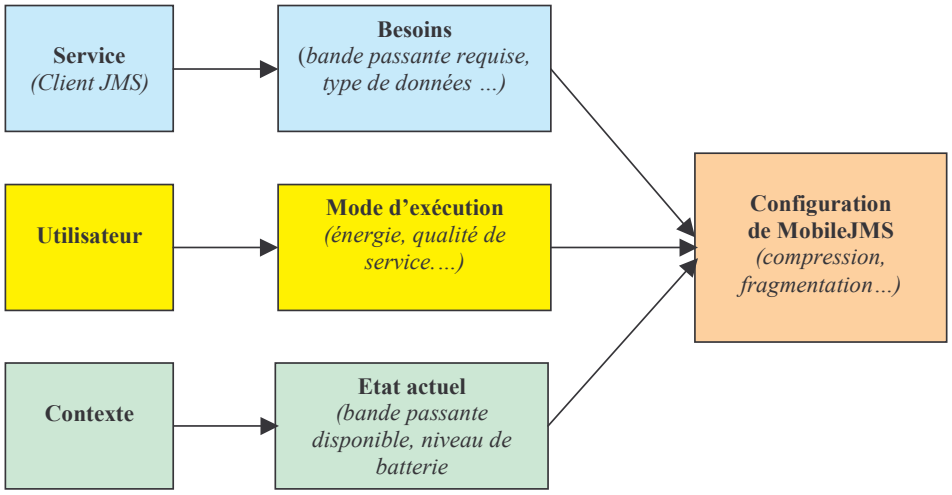


Fig. 6.1 – Relations entre les paramètres du modèle d'adaptation

6.2 Politiques d'adaptation

6.2.1 Définition

La politique d'adaptation constitue l'élément central dans le modèle d'adaptation. Elle consiste en un contrat qui associe un ensemble de besoins d'un service à un état du contexte d'exécution. Les politiques d'adaptation se divisent donc en deux parties : les besoins (*requirements*) et les contraintes (*constraints*). Les besoins spécifient les paramètres nécessaires à l'interaction

avec le service, comme le type de données utilisées et l'adresse ou le nom du serveur distant. Les contraintes spécifient les paramètres du contexte d'exécution qui doivent être respectés pour répondre à ces besoins (Par exemple, la bande passante nécessaire pour l'échange de messages ou la localisation géographique dans le cas d'un service géodépendant).

Une politique d'adaptation ne correspond qu'à un seul état du contexte. Pour faire face aux différents contextes que peut rencontrer un service, un ensemble de politiques d'adaptation lui est associé. La relation entre un service et les politiques d'adaptation peut s'exprimer de la manière suivante :

Un service = une interface + une implémentation + un ensemble de politiques d'adaptation

Les politiques sont définies par le fournisseur de service. Elle sont exprimées dans le langage XML. Elles adoptent la structure décrite dans la figure 6.2. L'adresse du serveur, qui fournit les données, dépend de la localisation géographique de l'utilisateur (latitude, longitude). Les types des données échangées avec le serveur dépendent de la bande passante disponible dans le réseau. Le type de réseau oblige le service à utiliser un réseau d'accès en particulier. Le délai maximum fixe la période pendant lequel le service peut s'exécuter en mode déconnecté (activation du S&F éventuellement).

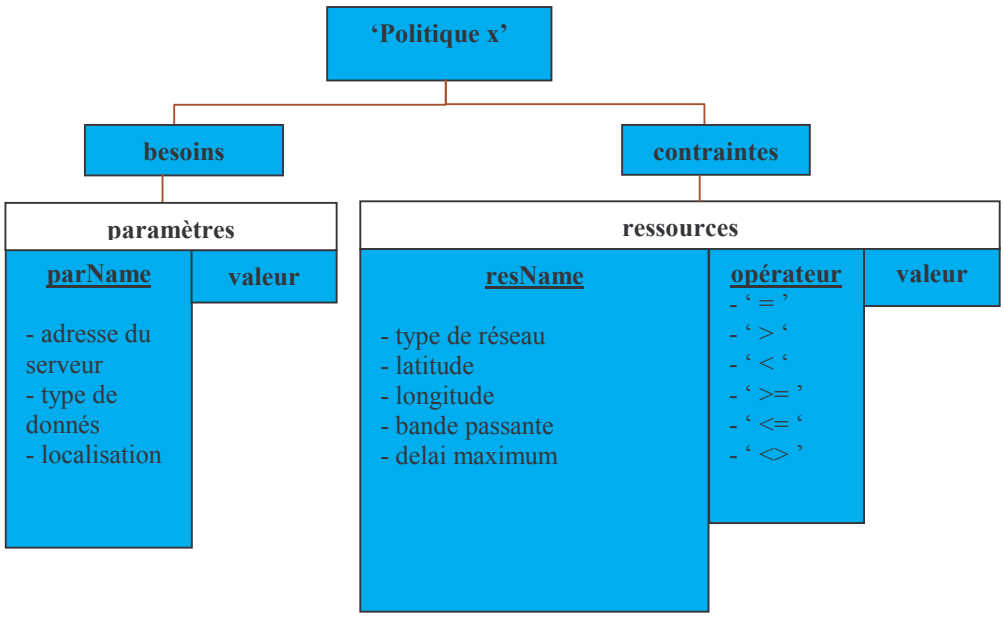


Fig. 6.2 – Structure de la politique d'adaptation

Ci-dessous, deux exemples de politiques d'adaptation, *HighBandwidth_ France* et *Medium-Bandwidth_ Germany*.

```
<policyE policyName="HighBandwidth_France">
    <requirements>
       <requirement parName="server"  value="Meteo_fr"/>
```

```
      <requirement parName="location"  value="France"/>
      <requirement parName="DataType"  value="Large_Image"/>
   <requirements>
   <constraints>
      <constraint resName="bandwidth" operator="greatherThan" value="200"/>
      <constraint resName="maximum_delay" operator="lessThan" value="5"/>
      <constraint resName="latitude" operator="greaterThan" value="110"/>
      <constraint resName="latitude" operator="lessThan" value="159"/>
      <constraint resName="longitude" operator="greaterThan" value="55"/>
      <constraint resName="longitude" operator="lessThan" value="100"/>
      <constraint resName="network" operator="equal " value="802.11b"/>
   </constraints>
```

Cette politique spécifie que les messages échangés par le service sont de type image haute définition, lorsque la bande passante, entre le client et le fournisseur, est supérieure à 200 kb/s. En outre, le serveur à contacter est *Meteo_fr*, lorsque l'utilisateur se trouve dans la zone géographique délimitée par les latitudes et les longitudes correspondant au territoire français.

```
<policyE policyName="MediumBandwidth_Germany">
      <requirements>
        <requirement parName="server"  value="Meteo_de" />
        <requirement parName="location"  value="Germany" />
        <requirement parName="DataType"  value="Small_Image" />
   <requirements>
   <constraints>
       <constraint resName="bandwidth" operator="greatherThan" value="120" />
       <constraint resName="maximum_delay" operator="lessThan" value="5" />
       <constraint resName="latitude" operator="greaterThan" value="162" />
       <constraint resName="latitude" operator="lessThan" value="190" />
       <constraint resName="longitude" operator="greaterThan" value="10" />
       <constraint resName="longitude" operator="lessThan" value="60" />
       <constraint resName="network"  operator="equal "value="802.11b" />
   </constraints>
```

Cette deuxième politique spécifie que les messages échangés sont de type image basse définition, lorsque la bande passante est supérieure à 120 kb/s. En outre, le serveur à contacter est *Meteo_ de*, lorsque l'utilisateur se trouve dans la zone géographique qui correspond au territoire allemand.

Il faut noter que les politiques d'adaptation ne spécifient pas le comportement que doit adopter l'intergiciel pour répondre aux besoins. Cette décision doit se prendre dynamiquement au moment de l'exécution, comme cela est décrit dans la section 6.5.

6.2.2 Représentation des politiques

Les politiques d'adaptation associées à un service donné sont structurées logiquement sous la forme d'un arbre. Cette structure facilite et optimise la recherche de la politique la plus adaptée au contexte d'exécution actuel. L'arbre est construit selon le principe de spécialisation : une politique donnée est un descendant d'une autre politique, si elle définit au moins les mêmes

besoins et contraintes. Afin de réduire le temps de recherche, les politiques sont organisées dans l'arborescence en partant de la contrainte la plus faible jusqu'à la contrainte la plus forte. Ceci implique que si la politique contenue dans le nœud courant, n'est pas adaptée au contexte, tous ses descendants ne le sont pas également. En outre, une politique ne peut pas avoir deux ascendants directs à la fois.

Les politiques qui possèdent en ensemble de contraintes non comparables, comme la bande passante et la localisation géographique, sont placées au même niveau de l'arborescence. En revanche, si une politique A spécialise la politique B, selon le critère de la bande passante, et qu'elle spécialise la politique C, selon le critère de la localisation géographique, la politique A pourrait être un descendant soit de B, soit de C.

La figure 6.3 représente l'exemple d'un arbre de spécialisation. Nous pouvons constater que la politique *HighBanwidth* est un descendant de la politique *NormalBandwidth*, car elle définit une contrainte de bande passante (bw>10) qui est incluse dans la contrainte définie par cette deuxième politique (bw>5). De la même manière, les politiques « Paris » et « Nice » sont des descendants de la politique « France », car ils spécialisent les coordonnées géographiques de la France.

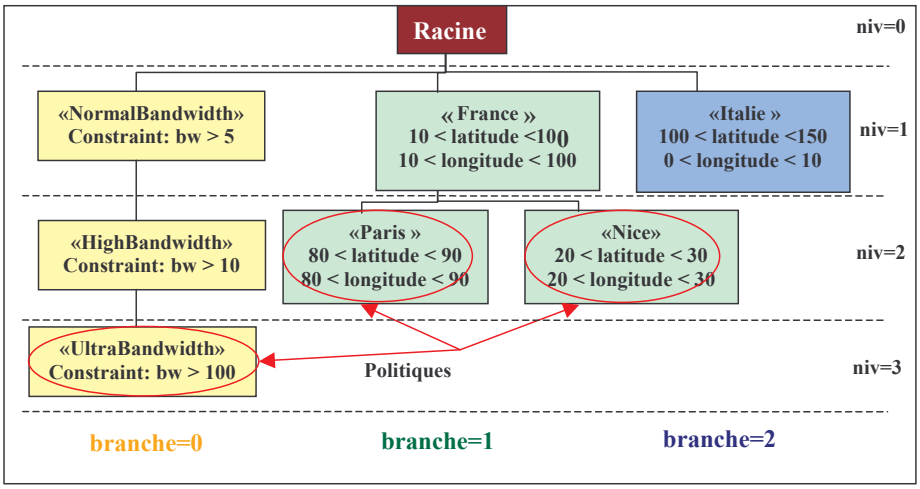


Fig. 6.3 – Exemple d'un arbre de spécialisation

6.3 Le gestionnaire d'adaptation

Le gestionnaire d'adaptation est le cadre dans lequel s'effectuent l'instanciation et la gestion des services (figure 6.4). Il inclut également toutes les opérations concernant l'analyse, le choix et la mise en œuvre des politiques d'adaptation des services. Ce module central relie les applications à l'intergiciel et au système d'exploitation. Il fait intervenir les entités suivantes : le gestionnaire de services, l'analyseur XML, le gestionnaire du contexte local (LCM), le gestionnaire de configuration et le coordinateur. Nous définissons ci-après le rôle de chaque entité.

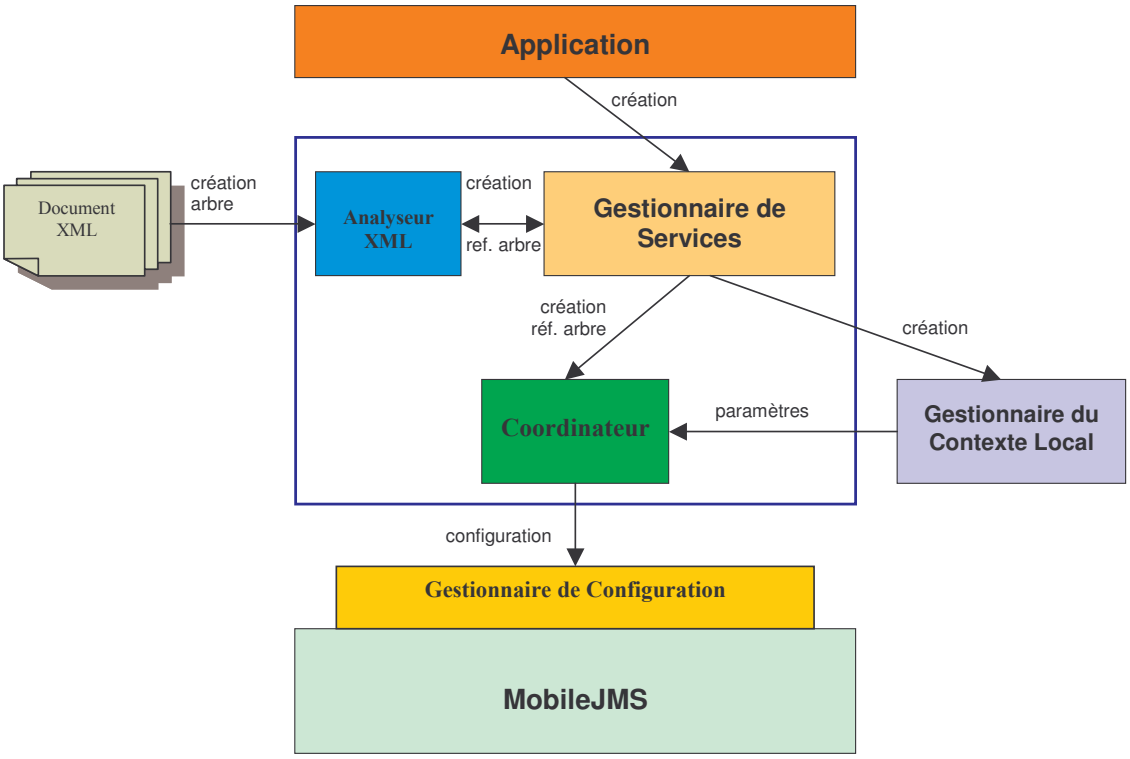


Fig. 6.4 – Architecture du gestionnaire d'adaptation

Le gestionnaire de services

Le gestionnaire de services est responsable de la gestion du cycle de vie des services. Il initialise, alloue les ressources nécessaires et ferme les services. Il reçoit comme argument un fichier XML qui décrit et paramètre les services utilisés par l'application.

Il est responsable également de la création de toutes les autres entités. Il fait appel notamment à l'analyseur XML pour la création d'un arbre de spécialisation qui correspond à chaque service. Il crée ensuite le coordinateur et lui fournit des références sur les différents arbres.

L'analyseur XML

L'analyseur XML est l'entité qui crée et gère l'arbre de spécialisation. Il transforme un ensemble de politiques d'adaptation, décrites dans un document XML, en objets Java. Ce module est basé sur le composant JAXB de Java [95] (figure 6.5) qui effectue les opérations suivantes :

- Générer et compiler des interfaces JAXB à partir d'un schéma XML décrivant la structure des politiques, puis créer des classes Java implémentant ces interfaces.
- Exécuter ces classes pour déballer (*unmarshall*), traiter et emballer (*marshall*) les docu-

ments XML à travers JAXB.

La première opération est effectuée uniquement au moment de la compilation de la plate-forme. Elle génère des classes Java à partir du schéma XML des politiques d'adaptation. En revanche, la deuxième opération est effectuée à chaque démarrage de la plate-forme. Elle permet d'analyser le document XML qui rassemble toutes les politiques d'adaptation associées au service, et de construire l'arbre ensuite.

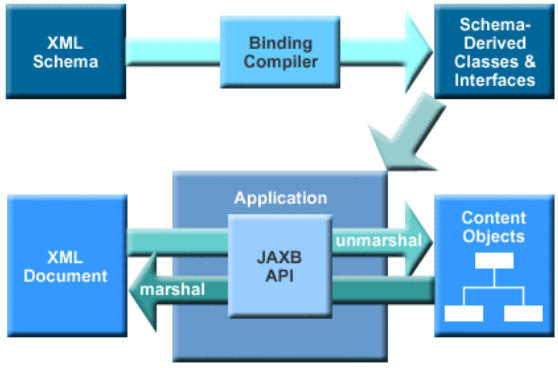


Fig. 6.5 – Analyse JAXB (src : java.sun.com/xml/jaxb)

Le gestionnaire du contexte local (LCM)

Le LCM maintient et met à jour les paramètres concernant le contexte d'exécution du terminal de l'utilisateur : type de réseau, bande passante, taux de perte de paquets, latence réseau, niveau de batteries, taux d'utilisation du CPU et le niveau de l'utilisation de la mémoire. Dans notre implémentation, ces paramètres sont simulés à l'aide d'un fichier XML. La modification de ce fichier peut se faire à l'aide d'une interface graphique ou à travers la ligne de commande.

Le coordinateur enregistre auprès du LCM les paramètres qu'il souhaite superviser. A chaque modification de l'un de ces paramètres, le LCM envoie une notification au coordinateur contenant la nouvelle valeur.

Le gestionnaire de configuration

Nous avons décrit précédemment cette entité dans la section 5.9. Le gestionnaire de configuration contrôle dynamiquement la configuration de MobileJMS (configuration du graphe de protocoles). Cette configuration est définie par le coordinateur.

Le coordinateur

Le coordinateur est l'entité centrale du gestionnaire d'adaptation. Sa principale fonction est de sélectionner la politique d'adaptation à appliquer. Il déclenche le processus de sélection à chaque fois qu'il y a un changement significatif du contexte d'exécution. Un tel changement de contexte se produit lorsque la valeur d'un paramètre du contexte se retrouve hors de l'intervalle défini lors de l'initialisation de la plate-forme. Il est notifié par le LCM.

Le coordinateur parcourt l'arbre de spécialisation et applique la politique qui fournit le meilleur bilan. Si p est une politique, c une configuration de MobileJMS et r un critère de performance :

$$Bilan = f(p, c, r)$$

Nous exposons dans la section suivante les relations entre les critères de performances et les services de compression et de fragmentation de MobileJMS. La définition de la fonction f fait l'objet de la section 6.5.

6.4 Modèle de performance de MobileJMS

Nous avons souligné, précédemment, qu'un service peut s'exécuter de différentes manières suivant le type de performance souhaité par l'utilisateur. Nous avons retenu dans notre modèle deux types de performance : la vitesse de transmission des messages et l'économie d'énergie.

La vitesse de transmission est un critère important, car plus elle est élevée, plus l'utilisateur pourra accéder rapidement à un contenu plus riche et utiliser le service de manière conviviale et réactive. C'est particulièrement le cas pour les services multimédia. La vitesse de transmission ne dépend pas seulement de la bande passante du réseau, mais également de la vitesse du traitement effectué sur la machine de l'utilisateur. L'énergie est également un critère de performance important pour les utilisateurs compte tenu de l'autonomie limitée des terminaux mobiles. Il s'agit donc d'étudier dans quelles conditions le service peut être utilisé de manière à minimiser l'usage d'énergie.

La mise en œuvre d'une politique d'adaptation associée à un critère de performance se traduit dans MobileJMS par la configuration du graphe de protocole utilisé. Dans ce cadre, le service de compression joue un rôle particulier. La compression des données permet d'appliquer des politiques qui nécessitent une bande passante supérieure à celle qui est disponible dans le contexte actuel. Elle réduit, en général, le temps nécessaire à la transmission des données sur le réseau en réduisant leur taille. Néanmoins, elle augmente, en même temps, le traitement nécessaire sur le terminal de l'utilisateur. La compression réduit le temps total d'envoi d'un message dans le cas suivant :

Le temps supplémentaire nécessaire à la compression sur le terminal est inférieur à la différence entre les temps de transmission sur le réseau du message non compressé et du message compressé.

Cette relation reste également valable si le critère pris en considération est la réduction de la consommation d'énergie.

L'emploi du service de fragmentation, associée à celui de la compression, permet également de réduire le temps de transmission et la consommation d'énergie, par rapport à l'emploi exclusif du service de compression dans les conditions que nous exposons dans les sous-sections suivantes.

La problématique qui nous est donc posée consiste à déterminer dans quel cas de figure l'usage de la compression et de la fragmentation améliore le type de performance souhaité par l'utilisateur. Nous avons conçu deux modèles qui permettent de prédire respectivement les temps de transmission et l'énergie consommée correspondant à chaque configuration de MobileJMS. Ces modèles sont utilisés dans l'estimation des bilans qu'effectue le coordinateur.

6.4.1 Prédiction des temps de transmission

Les formules de la compression

Le temps de transmission total d'un message non compressé peut être exprimé par la notation suivante :

$$t = \frac{s}{bw} \tag{6.1}$$

où s est la taille du message à transmettre par l'application et bw est la bande passante disponible sur la connexion réseau.

Le temps de transmission total d'un message compressé peut être exprimé par :

$$t_c = t_{comp} + t_{send} \tag{6.2}$$

où t_{comp} est le temps nécessaire à la compression et t_{send} est le temps nécessaire à la transmission sur le réseau du message compressé. Ces deux valeurs peuvent être estimées de la manière suivante :

$$t_{comp} = \frac{s}{cs} \tag{6.3}$$

$$t_{send} = \frac{s_{comp}}{bw} \tag{6.4}$$

$$s_{comp} = \frac{s}{ratio} \tag{6.5}$$

où cs est la vitesse de compression du message (nombre d'octets compressés par seconde), s_{comp} est le taille du message compressé et $ratio$ est le ratio de compression (le rapport entre la taille du message initial et la taille du message compressé).

Notons que cs inclut également le temps de copie du message compressé dans un tampon de la mémoire.

Par ailleurs, si un client reçoit un message compressé, le temps d'exécution total est :

$$t_d = t_{decomp} + t_{send} \tag{6.6}$$

$$t_{decomp} = \frac{s}{ds} \tag{6.7}$$

où t_{decomp} est le temps nécessaire à la décompression du message, t_{send} est estimé de la même manière que (6.4) et ds est la vitesse de décompression du message.

Les formules de la fragmentation

Dans les formules présentées précédemment, le message doit être entièrement compressé avant d'être transmis sur le réseau et doit être entièrement reçu avant de pouvoir être décompressé. La compression ou la décompression, d'un coté, et la transmission, de l'autre, se font à deux moments distincts.

Le découpage du message en plusieurs fragments permet d'entrelacer ces deux moments. La décompression peut commencer alors que le message n'a pas été entièrement reçu, car la réception du fragment *n+1* peut s'effectuer en même temps que la décompression du fragment *n*. Le service de fragmentation, utilisée au dessus du service de compression, permet donc de réduire le temps d'exécution dans le cas où il n'introduirait pas un temps supplémentaire très important (dû à la copie des données dans les tampons).

Le facteur déterminant dans le calcul du temps d'envoi ou de réception d'un message fragmenté et compressé est la différence entre le temps de traitement local et le temps de transmission. Si le temps de traitement local (temps de génération et de compression d'un fragment) est plus important que celui de transmission, alors le temps de traitement recouvre le temps de transmission et devient prépondérant dans le calcul de temps (figure 6.6a). Cette relation est inversée dans le cas contraire (figure 6.6b).

Le temps nécessaire à l'envoi d'un message fragmenté et compressé peut être exprimé par la formule suivante :

$$t_{comp\_frag} = \begin{cases} nt_{frag} + nt_{comp} + t_s & (t_{frag} + t_{comp} \geq t_s) \\ t_{frag} + t_{comp} + nt_s & (t_{frag} + t_{comp} < t_s) \end{cases} \tag{6.8}$$

où n est le nombre de fragments du message, t_{comp} est le temps de compression d'un fragment du message, t_{frag} est le temps nécessaire à la génération d'un fragment (copie dans un tampon de la mémoire) et t_s le temps de transmission correspondant sur le réseau.

Notons que nous ne prenons pas en considération, dans ce modèle, le temps de copie des données dans le tampon du connecteur réseau. Ce temps est beaucoup moins important que celui de la fragmentation, de la compression ou de la transmission.

De la même manière, le temps nécessaire à la réception d'un message fragmenté et compressé

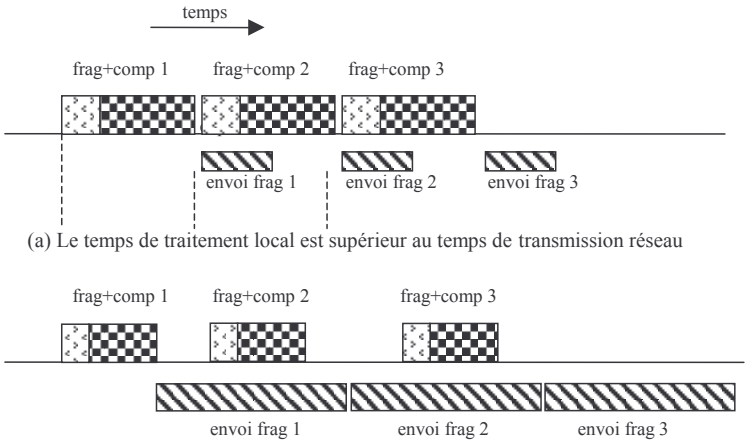


Fig. 6.6 – Envoi d'un message fragmenté et compressé

t_{defrag_decomp} s'exprime par :

$$t_{decomp\_defrag} = \begin{cases} nt_{defrag} + nt_{decomp} + t_s & (t_{defrag} + t_{decomp} \geq t_s) \\ t_{defrag} + t_{decomp} + nt_s & (t_{defrag} + t_{decomp} < t_s) \end{cases} \quad (6.9)$$

où n est le nombre de fragments du message, t_{defrag} est le temps de copie de chaque fragment et t_{decomp} est le temps nécessaire à la décompression d'un fragment.

Discussion

Les formules de compression et de fragmentation permettent de comparer les temps de transmission des messages selon la configuration utilisée par MobileJMS (sans compression, compression, fragmentation et compression). Elles peuvent donc être utilisées pour déterminer la configuration qui fournit la meilleure vitesse de transmission. Ces formules sont assez simples et dépendent de valeurs qui sont mesurables et relativement constantes. Elles n'introduisent pas donc une surcharge importante lors de l'évaluation des bilans. Notons que, lors de la transmission d'un message, nous ne considérons que les temps de compression et de décompression du côté du terminal mobile. Ces temps sont beaucoup moins importants du côté du fournisseur et possèdent donc moins d'influence sur les résultats.

Comme il est difficile d'estimer les différents paramètres de ces formules de manière théorique, nous avons effectué un ensemble de mesures sur le terminal cible en utilisant plusieurs types de compression et de données. Ces mesures fournissent des valeurs moyennes de ces paramètres. La section 8.2 décrit ces expérimentations.

6.4.2 Prédiction de la consommation d'énergie

Les formules de la compression

La compression et la décompression des données nécessitent généralement une importante utilisation du CPU et des accès intensifs à la mémoire. Ils impliquent donc une surconsommation de l'énergie des batteries de la machine. Toutefois, il est admis [5] que l'énergie nécessaire pour l'envoi d'un seul bit sur un réseau 802.11b est de l'ordre de 1000 fois plus importante que celle nécessaire à une seule opération de calcul en 32 bits. Par conséquent, si 1000 opérations de calcul peuvent réduire la taille des données, ne serait-ce que d'un seul bit, l'énergie peut être économisée. Néanmoins, l'accès intensif à la mémoire, qui caractérise les algorithmes de compression sans perte, peut être également coûteux de ce point de vue.

Les terminaux mobiles possèdent généralement des architectures matérielles simples avec un seul processeur. Le processeur est utilisé aussi bien pour le calcul que pour la communication réseau (écriture et lecture des données sur l'interface réseau, réassemblage de paquets, etc.). Le temps processeur est donc partagé entre les traitements locaux et les traitements réseau.

L'énergie utilisée par les interfaces réseau pour envoyer ou recevoir les données se divisent en deux parties : l'énergie pour envoyer ou recevoir les données sur le réseau et l'énergie consommée dans les intervalles séparant les instants d'arrivées des paquets (*idle*).

Dans le cas de la transmission d'un message non compressé de taille s (nombre de bits), l'énergie totale nécessaire peut s'exprimer par :

$$E = es + nt_{idle}p \tag{6.10}$$

où e est l'énergie nécessaire à l'envoi d'un seul bit, n est le nombre de paquets transmis, t_{idle} est le temps total des intervalles séparant deux transmissions successives de paquets de données et p est le niveau de tension pendant les temps de repos du CPU.

Le paramètre t_{idle} dépend de la bande passante actuelle du réseau. Plus la bande passante est faible, plus les intervalles entre les transmissions des paquets sont élevés. Le paramètre n peut être estimé approximativement par le rapport entre la taille du message et la taille d'un paquet. Nous supposons toutefois que la taille des paquets s_p est fixe, et nous ne prenons pas en considération les données de contrôle situées dans l'entête des paquets.

L'équation 6.10 peut alors s'écrire :

$$E = s(e + \frac{t_{idle}p}{s_p}) \tag{6.11}$$

Si le message est compressé, l'énergie nécessaire pour l'envoi du message s'exprime par :

$$E_{comp} = es_{comp} + t_{comp}p_{comp} + t_{idle}p\frac{s_{comp}}{s_p} \tag{6.12}$$

où s_p est la taille du message compressé, t_{comp} est le temps nécessaire à la compression et p_{comp} est le niveau de tension pendant le temps de compression du message.

En remplaçant s_{comp} et t_{comp} par les valeurs correspondantes des équations 6.3 et 6.5, on obtient :

$$E_{comp} = s(\frac{1}{ratio}(e + \frac{t_{idle}p}{s_p}) + \frac{p_{comp}}{cs}) \tag{6.13}$$

La transmission d'un message compressé permet donc d'économiser l'énergie si :

$$\boxed{E_{comp} < E \implies p_{comp} < cs(e + \frac{t_{idle}p}{s_p})(\frac{ratio-1}{ratio})} \tag{6.14}$$

où cs est la vitesse de compression et $ratio$ est le ratio de compression du message.

De la même manière, lors de la réception d'un message compressé de taille s, l'énergie totale nécessaire s'exprime par :

$$E_{decomp} = rs_{comp} + t_{decomp}p_{decomp} + t_{idle}p\frac{s_{comp}}{s_p} \tag{6.15}$$

ou

$$E_{decomp} = s(\frac{1}{ratio}(r + \frac{t_{idle}p}{s_p}) + \frac{p_{decomp}}{ds}) \tag{6.16}$$

où r est l'énergie nécessaire à la réception d'un seul bit, t_{decomp} est le temps nécessaire à la décompression, p_{decomp} est le niveau de tension moyen pendant la décompression, ds est la vitesse de décompression et $ratio$ est le ratio de compression du message.

La réception d'un message compressé permet donc d'économiser l'énergie si :

$$\boxed{E_{decomp} < E \implies p_{decomp} < ds(r + \frac{t_{idle}p}{s_p})(\frac{ratio-1}{ratio})} \tag{6.17}$$

Les formules de la fragmentation

Selon Xu [116], si les données sont reçues avec une bande passante de 602 ko/s, environ 30% de l'énergie totale de transmission est consommée pendant les moments qui séparent l'arrivée des paquets. Cette énergie, loin d'être négligeable, est consommée même si le processeur ne fait que recevoir des données. Ce constat nous amène à penser, que l'énergie serait mieux exploitée si ces moments étaient utilisés pour effectuer d'autres tâches, comme la compression ou la décompression des données.

Si le service de compression est utilisé tout seul, la décompression ne peut être déclenchée qu'après la réception de la totalité du message qui risque parfois d'être de taille importante. Dans le cas où il n'existerait pas d'autres processus qui s'exécutent en parallèle sur le processeur, les intervalles de temps qui séparent l'arrivée de paquets ne sont pas exploités pour effectuer d'autres tâches. Il est donc souhaitable que la décompression ne s'effectue pas d'un bloc (figure 6.7) sur

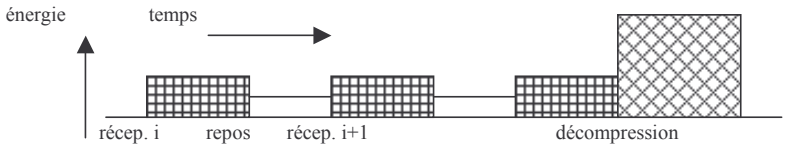


Fig. 6.7 – Consommation énergétique sans fragmentation

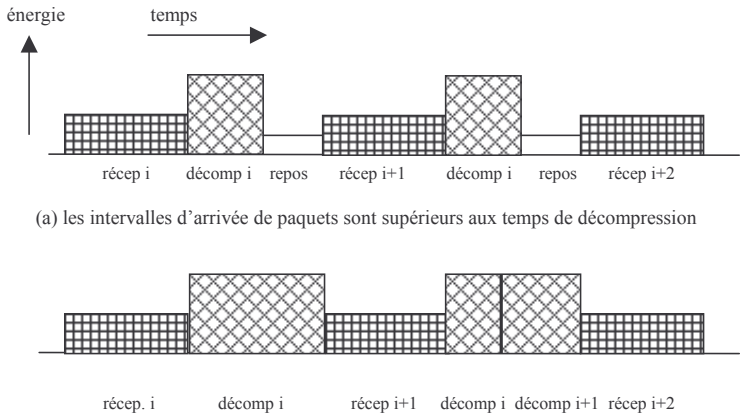


Fig. 6.8 – Consommation énergétique en utilisant la fragmentation

tout le message, mais sur des fragments du message. Ainsi, la décompression du fragment i peut se faire en même temps que la réception du fragment i+1.

On retrouve ici la configuration, présentée dans la section 6.4.1, utilisant les services de fragmentation et de compression. Il faut cependant que chaque fragment MobileJMS corresponde à un paquet de données au niveau réseau.

La figure 6.8 représente l'énergie consommée lors de la réception d'un message fragmenté et compressé. Sur la partie (a), on peut constater que les intervalles séparant les arrivées des paquets sont supérieurs aux temps nécessaires à la décompression. La décompression peut s'effectuer entièrement pendant ces intervalles. Les seuls intervalles de repos, dans ce cas, correspondent à la différence entre le total des intervalles d'arrivés des paquets et le total des temps de décompression. En revanche, sur la partie (b), les intervalles séparant les arrivées des paquets sont inférieurs aux temps de décompression. Dans ce cas, tous ces intervalles sont utilisés pour la décompression. Ce qui signifie que l'énergie est utilisée de manière optimale.

Notons que nous ne prenons pas en considération, dans ce modèle, l'énergie nécessaire à la fragmentation, car elle est beaucoup moins importante que celle nécessaire pour la compression ou la transmission. Nous supposons également que le temps de compression (décompression) de tout le message est égal au total des temps de compression (décompression) des fragments.

Si le temps séparant l'arrivée de deux paquets t_{idle} est supérieur au temps nécessaire à la compression d'un fragment t_{cf} ($t_{cf} = \frac{t_{comp}}{n}$, où n est le nombre de fragments et t_{comp} est le temps utilisé dans 6.12), l'énergie nécessaire à l'envoi d'un message fragmenté et compressé de taille s s'exprime par :

$$E_{comp\_frag} = es_{comp} + t_{comp}p_{comp} + (t_{idle} - t_{cf})p\frac{s_{comp}}{s_p} \tag{6.18}$$

$$E_{comp\_frag} = s(\frac{1}{ratio}(e + \frac{t_{idle}p}{s_p} - \frac{p}{cs}) + \frac{p_{comp}}{cs}) \tag{6.19}$$

Si le temps séparant l'arrivée de deux paquets est inférieur au temps nécessaire à la compression d'un fragment, l'énergie nécessaire à l'envoi d'un message fragmenté et compressé de taille s s'exprime par :

$$E_{comp\_frag} = es_{comp} + t_{comp}p_{comp} \tag{6.20}$$

$$E_{comp\_frag} = s(\frac{e}{ratio} + \frac{p_{comp}}{cs}) \tag{6.21}$$

où e est l'énergie nécessaire à la réception d'un seul bit, p_{comp} est le niveau de tension moyen pendant la compression, cs est la vitesse de compression et $ratio$ est le ratio de compression du message.

De la même manière, nous pouvons exprimer l'énergie nécessaire à la réception d'un message fragmenté et compressé par les formules suivantes :

Si $t_{idle} > t_{df}$ (t_{df} est le temps nécessaire à la décompression d'un fragment) :

$$E_{decomp\_defrag} = rs_{comp} + t_{decomp}p_{decomp} + (t_{idle} - t_{df})p\frac{s_{comp}}{s_p} \tag{6.22}$$

$$E_{decomp\_defrag} = s(\frac{1}{ratio}(r + \frac{t_{idle}p}{s_p} - \frac{p}{ds}) + \frac{p_{decomp}}{ds}) \tag{6.23}$$

Si $t_{idle} \leq t_{df}$

$$E_{decomp\_defrag} = rs_{comp} + t_{decomp}p_{decomp} \tag{6.24}$$

$$E_{decomp\_defrag} = s(\frac{r}{ratio} + \frac{p_{decomp}}{ds}) \tag{6.25}$$

où r est l'énergie nécessaire à la réception d'un seul bit, p_{decomp} est le niveau de tension moyen pendant la décompression, ds est la vitesse de compression et $ratio$ est le ratio de compression du message.

Discussion

Nous remarquons qu'aussi bien dans le cas de la compression (6.18 et 6.20), que celui de la décompression (6.22 et 6.24), l'énergie nécessaire à l'envoi ou la réception d'un message est inférieure à l'énergie nécessaire à l'envoi ou à la réception de ce message en n'utilisant que la compression (6.13 et 6.15). Cela signifie que la fragmentation associée à la compression permet un gain d'énergie dans tous les cas par rapport à l'utilisation que de la compression.

Notons que les paramètres nécessaires pour l'estimation de l'énergie dépendent du type des données (ratio), du type de la machine et de l'interface réseau (e, r, p, p_{comp}, p_{decomp}) et du type des données et de la machine (cs, ds). Nous pouvons remarquer que plus les paramètres $ratio$, cs et ds sont élevés, moins l'énergie nécessaire à l'envoi des messages est élevée et plus il y a de gain par rapport à l'envoi d'un message sans compression.

On retrouve ici l'opposition classique entre le ratio, qui est maximisé en utilisant les taux de compression les plus élevés, et la vitesse de compression qui est maximisée en utilisant les plus faibles taux de compression. Par conséquent, la compression la plus efficace, du point de vue de l'énergie, ne peut être déduite uniquement à partir de son taux. Cela nous suggère d'estimer les gains générés par tous les types de compression dans l'estimation des bilans.

De la même manière que pour le modèle de prédiction des temps de transmission, les paramètres de ce modèle peuvent être mesurés de manière expérimentale. Les résultats resteront néanmoins très dépendants du type de la machine et du réseau. Nous n'avons pas traité dans le cadre de ce travail l'estimation de ces paramètres. Nous nous sommes contentés de présenter un modèle théorique qui permet de montrer la pertinence de l'utilisation des services de compression et de fragmentation dans le but de réduire l'énergie consommée.

6.5 La prise de décision

La prise de décision concerne la recherche de la politique d'adaptation qui fournit le meilleur bilan. L'estimation des bilans de chaque politique prend en considération une certaine configuration de MobileJMS et le mode d'exécution souhaité par l'utilisateur. Nous avons défini deux modes d'exécution d'un service :

- **Le mode de qualité de service :** dans ce mode, la politique choisie est celle qui fournit la meilleure qualité de service possible par rapport au contexte d'exécution. Cette qualité de service se traduit par l'application de la politique qui réclame la bande passante la plus élevée, ce qui signifie des données de meilleure qualité.
- **Le mode d'économie d'énergie :** dans ce mode, la politique choisie est celle qui effectue le meilleur compromis entre la qualité des données utilisées par le service et la consommation d'énergie.

Le premier critère qui permet d'évaluer une politique est celui de la localisation géographique. Si les coordonnées actuelles de l'utilisateur n'appartiennent pas à l'espace géographique défini dans la politique P, celle-ci n'est pas applicable. Si la politique P est applicable, le coordina-

teur procède à l'évaluation de son bilan par rapport à toutes les configurations possibles de MobileJMS. Ces configurations sont :

- Configuration de base : sans service à valeur ajoutée.
- Compression ZIP de niveau 1 (faible).
- Compression ZIP de niveau 4 (moyenne).
- Compression ZIP de niveau 9 (forte).
- Compression GZIP (un seul niveau est prédéfini).
- Fragmentation avec l'une des couches de compression citées précédemment.

Si deux politiques possèdent un même bilan, la politique retenue est celle qui est la plus spécialisée. Par exemple, la politique qui indique Paris comme zone géographique est plus spécialisée que celle qui indique la France.

6.5.1 Le mode de qualité de service

Mis à part la localisation géographique, les deux autres critères qui permettent d'évaluer les bilans d'une politique dans ce mode, sont la bande passante de la politique et la vitesse de transmission. La politique retenue est celle qui fournit le plus grand bilan, en sachant qu'une même politique peut fournir plusieurs bilans suivant la configuration C de MobileJMS qui est appliquée.

Les bilans d'une politique P sont évalués à l'aide de l'algorithme suivant :

```
Si (position_actuelle ∈ P.zone_geographique)
   // Pour toutes les N configurations possibles de MobileJMS
      Pour i de 1 à N faire
           Si v(config(i)) ≥ P.bw
              Bilan(i) = P.bw × v(config(i))
           Sinon
              Bilan = 0 // Configuration non applicable
Sinon
   Bilan = 0 //Politique non applicable
```

Où,

- *v(config(i))* représente la vitesse de transmission des messages. Elle est estimée par rapport à la configuration *config(i)* et la bande passante actuelle du réseau. Cette estimation est basée sur les formules du modèle de prédiction des temps de transmission (section 6.4.1) ;
- *P.bw* représente la bande passante requise par la politique P.

La vitesse de transmission doit être supérieure à la bande passante spécifiée dans la politique pour que celle-ci soit applicable. Une vitesse inférieure signifierait que MobileJMS ne peut pas répondre, avec cette configuration, à la bande passante réclamée par le service.

La recherche du meilleur bilan se fait donc selon deux axes : l'axe des politiques et l'axe des configurations MobileJMS. L'évaluation des bilans s'effectue en multipliant la vitesse de transmission par la bande passante de la politique. Cette formule favorise les politiques qui réclament une meilleure bande passante, dans le cas où la vitesse de transmission permettrait l'application de plusieurs politiques. Elle garantit donc la meilleure qualité de service possible.

Nous présentons dans la section 8.3 quelques exemples de l'estimation des bilans dans ce mode.

6.5.2 Le mode d'économie d'énergie

Dans ce mode d'exécution, un quatrième critère est utilisé pour l'évaluation du bilan : la consommation de l'énergie d'énergie.

Les bilans d'une politique P sont évalués à l'aide de l'algorithme suivant :

```
Si (position_actuelle ∈ P.zone_geographique)
   // Pour toutes les N configurations possibles de MobileJMS
      Pour i de 1 à N faire
            Si v(config(i)) ≥ P.bw
               Bilan(i) = P.bw / energie(config(i))
            Sinon
               Bilan = 0 // Configuration non applicable
Sinon
   Bilan = 0 //Politique non applicable
```

Où,

- *P.bw* et *v(config(i))* représentent les mêmes paramètres utilisés dans le mode de qualité de service ;
- *energie(config(i))* est l'énergie par seconde, nécessaire pour assurer la transmission des messages suivant la configuration *config(i)* de MobileJMS. Cette énergie est calculée par les formules présentées dans le modèle de prédiction de consommation d'énergie (6.4.2).

Comme dans le mode de qualité de service, la vitesse de transmission d'une configuration doit être supérieure à la bande passante réclamée par la politique. En revanche, les bilans sont évalués par le rapport entre la bande passante de la politique et l'énergie nécessaire pour l'exécution. La politique retenue est celle qui effectue le meilleur compromis entre la bande passante qu'elle réclame et l'énergie nécessaire à sa mise en oeuvre.

6.5.3 L'utilisation du service de stockage et ré-émission (S&F)

La prise de décision concernant l'utilisation du service S&F (section 5.4.1) se déroule de manière indépendante de l'estimation des bilans et donc des modes d'exécution. Le S&F est géré de la manière suivante :

- si le coordinateur obtient une notification de la déconnexion réseau du client et qu'un délai d'attente maximum est spécifié dans la politique d'adaptation actuelle, le S&F est activé et rajouté au graphe de protocoles. Si aucun délai n'est précisé, le coordinateur décide de déconnecter l'application JMS définitivement du fournisseur JMS (fermeture de tous les objets JMS) ;
- si ce délai expire avant la reconnexion du client, le service JMS est définitivement déconnecté ;
- si le client se reconnecte avant l'expiration de ce délai, le coordinateur désactive le S&F après avoir envoyé tous les messages qui étaient en attente.

6.6 Synthèse

Ce chapitre nous a permis de compléter l'approche d'adaptation, qui est fondamentale dans les environnements mobiles, par la définition et la mise en œuvre d'un modèle d'adaptation. Ce modèle fait coopérer plusieurs niveaux d'un système mobile : utilisateurs, applications, intergiciel. Il est basé sur le concept de politique d'adaptation qui définit une relation entre les besoins des applications et les contraintes du contexte d'exécution. Nous avons proposé des algorithmes qui permettent de déterminer la politique la plus adaptée suivant l'état du contexte et les critères de performance recherchés par l'utilisateur.

Bien que l'implémentation soit liée aux propriétés spécifiques de MobileJMS, nous pensons que les concepts essentiels de ce modèle demeurent valables pour d'autres types d'applications et d'intergiciels. En particulier, le concept de politique d'adaptation ne dépend pas du fonctionnement de MobileJMS. Il fournit juste un cadre que l'intergiciel doit prendre en considération dans son comportement. Nous avons entrepris des améliorations qui visent à introduire la possibilité de modifier et de mettre à jour les politiques pendant l'exécution des services.

Nous pensons également que l'utilisateur doit intervenir dans la prise de certaines décisions. L'équation entre les besoins des services, la configuration de l'intergiciel et le contexte d'exécution, ne peut être complète qu'à travers les préférences des utilisateurs. L'utilisateur peut déterminer la hiérarchie des critères de performance qu'il souhaite avantager. Ainsi, nous avons introduit le mode d'exécution qui permet de moduler la manière dont le modèle traduit les besoins, définis dans les politiques, en une configuration de l'intergiciel.

Nous avons élaboré un modèle qui relie certains critères de performance aux configurations de MobileJMS. Nous avons constaté que les services de compression et de fragmentation peuvent tenir un rôle important pour l'amélioration de la qualité de service et la réduction de la consommation d'énergie. Une partie du chapitre 8 nous permettra de juger de la pertinence de ce modèle en opposant les valeurs théoriques avec celles mesurées lors de l'exécution de la plate-forme. No-

tons enfin que ce modèle peut être généralisé afin d'inclure d'autres critères de performance, comme l'utilisation des ressources locales de la machine (CPU, mémoire, etc.).

Chapitre 7

Vérification formelle des mécanismes de négociation et de reconnexion

Nous avons présenté dans le chapitre 5 les mécanismes de négociation dynamique et de reconnexion transparente de MobileJMS. Ces deux mécanismes impliquent des séquences d'échange de messages entre le client et le fournisseur. Ces échanges peuvent se dérouler en même temps que ceux des données ; il est donc possible que les séquences de négociation et de reconnexion soient entrelacées. A cela il faut rajouter que les processus qui génèrent ces échanges s'exécutent de manière concurrente au niveau des clients et du fournisseur JMS. Cette partie de l'intergiciel nécessite donc une vérification formelle de ses propriétés comportementales, comme l'absence de blocage, l'absence de cycles de non progression, la vivacité ou l'atteignabilité.

Ce chapitre présente les outils et les techniques déployés pour vérifier les propriétés comportementales de ces deux mécanismes. Il existe généralement deux familles de méthodes formelles : la vérification basée sur le raffinement et la preuve (par exemple, B [21] ou Z [1]) et la vérification de modèle (par exemple, PROMELA [37] ou LUSTRE [34]). Dans la première famille, le modèle représentant le système à vérifier est décrit à travers des axiomes. Les propriétés sont des théorèmes qui doivent être vérifiés en utilisant un prouveur de théorème. Dans la deuxième famille, le modèle est exprimé avec un langage à partir duquel une sélection exhaustive de l'espace d'état du modèle peut être déduite. Cet espace d'état est un graphe où les actions (instructions atomiques du langage) sont représentées par des états (une certaine valeur du contexte du système). Un outil réservé à cet usage permet généralement d'explorer ce graphe pour vérifier que les propriétés recherchées sont bien valides.

Nous avons opté pour la vérification de modèles car elle est destinée aux systèmes à états finis et fournit des outils automatiques pour la simulation et la vérification des propriétés de l'espace d'état. Nous avons utilisé le langage PROMELA qui permet de décrire des processus concurrents et qui fournit des canaux de communication entre processus. L'outil SPIN [38] permet d'interpréter le langage PROMELA et d'effectuer des simulations et des vérifications.

Nous présentons d'abord la vérification de modèles et le langage PROMELA. Nous traitons ensuite les machines d'états finis qui transposent le comportement de MobileJMS, puis nous présentons les résultats des vérifications effectuées par PROMELA/SPIN.

7.1 Vérification de modèles

7.1.1 Définition

La vérification de modèles (*model checking*) est une technique de vérification de systèmes concurrents à états finis. Cette technique a obtenu un grand succès dans le domaine de conception de systèmes matériels ; elle commence à obtenir également des résultats intéressants dans le domaine de la vérification des systèmes logiciels. Il s'agit de tester algorithmiquement si un modèle donné, le système lui-même ou une abstraction du système, satisfait une spécification logique (propriété), généralement formulée en terme de logique temporelle. Des algorithmes symboliques sont utilisés pour traverser le modèle, représentant un système donné, afin de vérifier si toutes les configurations (états) possibles valident les propriétés. L'ensemble de toutes ces configurations est appelé l'espace d'états. Si cet espace est fini, ces algorithmes peuvent être utilisés pour obtenir une preuve automatique. La violation d'une propriété implique la découverte d'un contre-exemple.

L'un des inconvénients majeurs à la vérification de modèles demeure le problème de l'explosion d'états. En effet, la taille du graphe de transitions croit exponentiellement alors que la taille du système ne croit que linéairement. La conséquence immédiate est que l'exploration exhaustive du graphe, qui est généré à partir d'un système complexe, est souvent impraticable.

Le modèle d'un système est décrit généralement par une spécification écrite dans un langage formel. Nous décrirons ci-après l'un de ces langages : PROMELA.

7.1.2 Le langage PROMELA

PROMELA [37] est un langage de spécification de systèmes concurrents asynchrones. Il permet de décrire des systèmes concurrents, comme les protocoles réseau, les systèmes de téléphonie, ou les programmes qui communiquent via des variables partagées ou le passage de messages synchrone/asynchrone. Ce langage autorise la création dynamique de processus. Les communications peuvent s'effectuer de différentes manières : partage de variables globales, envoi et réception de messages via des canaux de communication. Ces derniers peuvent être définis pour réaliser des communications tant synchrones (rendez-vous) qu'asynchrones (utilisation de tampons).

Le langage PROMELA utilise des types de données simples (*short, byte, bit, etc.*), des canaux de communication (ou *channel* entre processus), des structures de contrôle (*if..fi, do..od, etc.*), des instructions gardées, des expressions arithmétiques et logiques, et des primitives de lecture/écriture sur les canaux. Une spécification PROMELA est constituée de processus, de variables et de canaux de communication servant à transmettre des messages. Les canaux et les variables peuvent être déclarés soit globalement, soit localement à l'intérieur d'un processus. Les processus spécifient le comportement du système alors que les canaux et les variables définissent l'environnement d'exécution.

En PROMELA, il n'y a aucune distinction entre les conditions et les instructions. L'exécution d'une instruction ne peut avoir lieu que si celle-ci est exécutable. Une instruction est soit exécutable, soit bloquée. Une condition ne peut être passée que si elle est satisfaite. Si ce n'est pas

le cas, elle est bloquée jusqu'à ce qu'elle devienne vraie. Ci-dessous, un exemple d'un processus PROMELA :

```
Proctype counter() {     do
      ::  count = count +1
      ::  count = count -1
      ::  (count == 0) -> break
      od
}
```

Ce processus décrit un comportement répétitif (`do..od`) dans lequel il y a trois actions. Les deux premières actions d'incrémentation et de décrémentation peuvent s'exécuter dans un ordre aléatoire. La dernière est une action gardée (`count==0`). Cette garde spécifie une condition qui doit être satisfaite avant que l'action, située après, ne puisse s'exécuter (`break`).

7.1.3 La plate-forme SPIN

La vérification de la correction d'un système décrit en PROMELA peut être réalisée grâce à l'outil de simulation et d'analyse SPIN [38]. SPIN simule l'exécution du système en faisant des choix aléatoires concernant les ordres d'exécution des différents processus. Il permet également de générer un programme qui effectue une validation exhaustive par construction du graphe d'états. Lors des simulations et des validations, SPIN peut vérifier les propriétés définies par la spécification PROMELA. Ces propriétés sont :

- **L'absence de blocage :** le blocage se produit quand le système atteint un état final qui n'a pas été spécifié comme tel par le modèle PROMELA. Ce modèle doit spécifier les états finaux appropriés qui se caractérisent par le fait que tous les processus instanciés soient terminés et que tous les canaux de communication soient vides.
- **L'absence de cycles de non-progression :** un état est étiqueté comme état de progression, s'il doit être nécessairement exécuté par le système pour progresser. Un exemple est l'incrémentation du numéro de séquence dans l'envoi de message. Les séquences d'exécution, qui n'atteignent jamais ces états, sont appelées les cycles de non-progression.
- **La vivacité :** cette propriété est l'opposée de celle de l'absence de non-progression. La vivacité implique qu'un état donné ne peut être infiniment visité dans une séquence d'exécution donnée.
- **Les propriétés temporelles :** ces propriétés dérivent de l'expression suivante : « chaque état dans lequel la propriété P est vraie doit être suivi d'un état dans lequel la propriété Q est vraie ».

Par ailleurs, SPIN permet de détecter les réceptions non spécifiées de messages ainsi que les parties de code qui ne sont jamais atteintes, ni par conséquent exécutées (code mort). SPIN peut également être utilisé pour vérifier des invariants du système (spécifié par le mot-clé `assert` de PROMELA).

Le système de vérification de SPIN est basé sur le paradigme de la vérification à la volée. Cette vérification s'appuie sur une technique de réduction utilisant des ordres partiels. Des techniques de compression de la représentation en mémoire du graphe d'états peuvent être utilisées pour permettre la construction de graphes de taille très importante. Ce graphe d'états est codé sous la forme d'un vecteur, où chaque état comporte la valeur de toutes les variables ainsi que les compteurs de programme (la position actuelle de l'exécution) des processus instanciés.

Nous pouvons citer, en particulier, la technique du *« Bitstate hashing »* [37] qui est l'une des plus efficaces en matière de compression. L'idée de base est d'utiliser une table de hachage permettant d'accélérer la recherche dans la phase de comparaison pour effectuer la compression. Un état est codé sur un bit puisque pour chaque état est calculé une valeur de la fonction de hachage. La position correspondante dans la table est marquée de manière à indiquer que l'élément existe déjà.

L'inconvénient de cette technique est que la fonction de hachage n'est pas bijective et il est donc possible que deux états différents possèdent la même valeur de hachage. Dans ce cas, ils seront confondus et une partie de l'espace d'états ne sera probablement jamais explorée. Cette technique ne garantit pas donc l'absence totale d'erreurs, mais permet néanmoins d'aborder des modèles d'une complexité telle qu'une recherche exhaustive ne peut traiter.

7.2 Vérification. dans MobileJMS

7.2.1 Approche

Les mécanismes de négociation et de reconnexion permettent d'optimiser les performances de MobileJMS. Ils génèrent cependant un nombre important de séquences d'exécution car leur déclenchement et leur déroulement sont indépendants l'un de l'autre. Pour que le comportement de la plate-forme soit conforme à la spécification, chaque mécanisme ne doit pas annuler ou influer sur le bon déroulement de l'autre. Si une phase de reconnexion est déclenchée, elle ne doit pas annuler une phase de négociation qui a été entreprise avant la déconnexion. Ceci est nécessaire car si le graphe de protocoles à été modifié d'un côté (client ou fournisseur), il doit l'être également de l'autre. Dans le cas contraire, des messages JMS risquent d'être mal interprétés lors de la réception. Par ailleurs, il faut également s'assurer que tous les messages JMS arrivent dans le bon ordre même durant des phases de négociation ou de reconnexion.

Le débogage de la plate-forme ne permet pas de vérifier le bon déroulement de toutes les séquences d'exécution car il est pratiquement impossible de les reproduire totalement. Nous avons donc jugé utile de procéder à une vérification formelle de cette partie de MobileJMS. Cette vérification nous permettra notamment de tester les propriétés suivantes :

- L'absence d'interblocages (*deadlocks*).
- L'absence de cycles de non-progression.
- La vivacité.
- Pas de réception non spécifiées ou inattendues de messages.
- Toutes les parties du code sont atteintes.
- Le bon déroulement de la négociation pendant les phases de déconnexion.

– La bonne transmission des messages pendant les phases de négociation et de déconnexion.

La spécification et la vérification de ces propriétés se déroulent en trois parties :
– Définir des machines d'états finis (FSM) qui reflètent le comportement de MobileJMS.
– Ecrire une spécification PROMELA à partir de ces machines.
– Simuler et vérifier cette spécification à l'aide de SPIN.

Afin d'éviter le problème de l'explosion combinatoire, l'abstraction est l'une des approches les plus suggérées dans la littérature [85]. Par conséquent, nous avons simplifié autant que possible la spécification formelle représentant l'implémentation de MobileJMS. Nous avons supprimé de la spécification PROMELA toutes les informations qui ne sont pas strictement nécessaires pour démontrer les propriétés que nous avons citées ci-dessus. Ainsi, la représentation du graphe de protocoles, qui est susceptible de générer un très grand nombre d'états, a été réduite à une couche supérieure JMS, à une seule couche intermédiaire, à la couche d'aiguillage et à un connecteur.

7.2.2 Formalisation des mécanismes avec les FSMs

Une machine d'états finis (FSM) permet d'obtenir une représentation formelle du comportement de la plate-forme. Nous avons conçu deux machines décrivant respectivement le comportement du client et du fournisseur. Elles représentent une connexion JMS ouverte entre les deux entités. La FSM du fournisseur ne gère que l'interaction avec un seul client à la fois.

Les transitions d'un état vers un autre sont étiquetées suivant la notation suivante :

$$\frac{input, condition}{output, action}$$

où
– *input* représente la réception d'un message.
– *condition* est une condition logique qui doit être vérifiée pour effectuer la transition.
– *output* représente l'envoi d'un message.
– *action* est une action qui met à jour les informations sur l'état local.

Les types de messages que nous avons considérés sont :

– C_MSG : message JMS envoyé par le client au fournisseur.
– P_MSG : message JMS envoyé par le fournisseur au client.
– $CFG\_Request$: demande d'une nouvelle négociation.
– CFG_Accept : acceptation d'une demande de négociation.
– CFG_Reject : rejet d'une demande de négociation.
– CFG_Conf : confirmation de la demande de négociation.
– $RECONNECT\_Request$: demande d'une reconnexion.
– $RECONNECT\_Accept$: acceptation d'une demande de reconnexion.
– $RECONNECT\_Reject$: rejet d'une demande de reconnexion.
– $RECONNECT\_Conf$: confirmation de la demande de reconnexion.

- $DISCONNECTION$: message indiquant une déconnexion réseau. Il est envoyé par le connecteur à la couche supérieure au niveau du client ou du fournisseur.
- $RECONNECTION$: message indiquant qu'une nouvelle connexion réseau est disponible.
- ACK : confirmation de l'envoi du message par le connecteur.

Les variables utilisées par le client et le fournisseur sont :
- msg_id : identificateur utilisé par le client pour étiqueter le prochain message à envoyer ;
- $last\_msg\_id$: identificateur utilisé dans l'envoi du message précédent ;
- $last\_received\_msg\_id$: identificateur du dernier message reçu ;
- $last\_received\_cfg\_id$: identificateur du dernier message de négociation reçu ;
- $sending$, $connected$, $config\_state$: booléens indiquant, respectivement, si l'envoi de messages est actif, si le client est connecté et s'il est dans l'état de négociation.

FSM du Client

Le FSM du client est représenté sur la figure 7.1, où :

- C1 : $sending == true$; $config\_state == false$ ($==$ est une condition d'égalité) ;
- C2 : $P\_MSG.msg\_id == last\_received\_msg\_id + 1$.

et
- A1 : $last\_msg\_id = msg\_id$; msg_id++ ; $C\_MSG.msg\_id = msg\_id$;
- A2 : $sending = false$; $config\_state = true$; $last\_cfg\_id = cfg\_id$; cfg_id++ ; $cfg\_req.msg\_id = cfg\_id$;
- A3 : $sending = true$; $config\_state = false$;
- A4 : reconfiguration du graphe de protocoles ; $sending = true$; $config\_state = false$;
- A5 : $last\_received\_msg\_id++$;
- A6 : $sending = false$; fermeture du connecteur ;
- A7 : $sending = true$; réassociation avec le reste du graphe ;
- A8 : fermeture du graphe de protocoles et fermeture de la connexion JMS.

Cette machine possède un état initial ($working$) et deux états finaux ($working$, $final$). A partir de l'état initial, le client peut envoyer un message JMS (C_MSG) au fournisseur. Ce message contient un identificateur que le client incrémente lors de l'envoi de chaque nouveau message, il doit être acquitté par le connecteur de transport (état 2) par un message ACK. Le client retourne à l'état initial si l'état de configuration est faux ($config\_state == false$). Ceci empêche le déclenchement d'une négociation si une autre négociation est déjà en cours. A l'état initial, le client peut également recevoir un message JMS du fournisseur (P_MSG). Mais il doit vérifier que l'identificateur contenu dans ce message suit dans l'ordre l'identificateur du message qu'il a reçu précédemment (condition C2).

Par ailleurs, le client peut également envoyer, à partir de l'état initial, une demande de négociation ($CFG\_Request$) à condition que l'état de configuration soit faux. Il arrête l'envoi des autres messages et met l'état de négociation à vrai (action A2). L'acquittement de cette demande par l'état 2 fait passer le client à l'état 3 où il se met en attente de la réponse du fournisseur. Si le client reçoit une réponse positive (CFG_Accept), il envoie un message de confirmation au

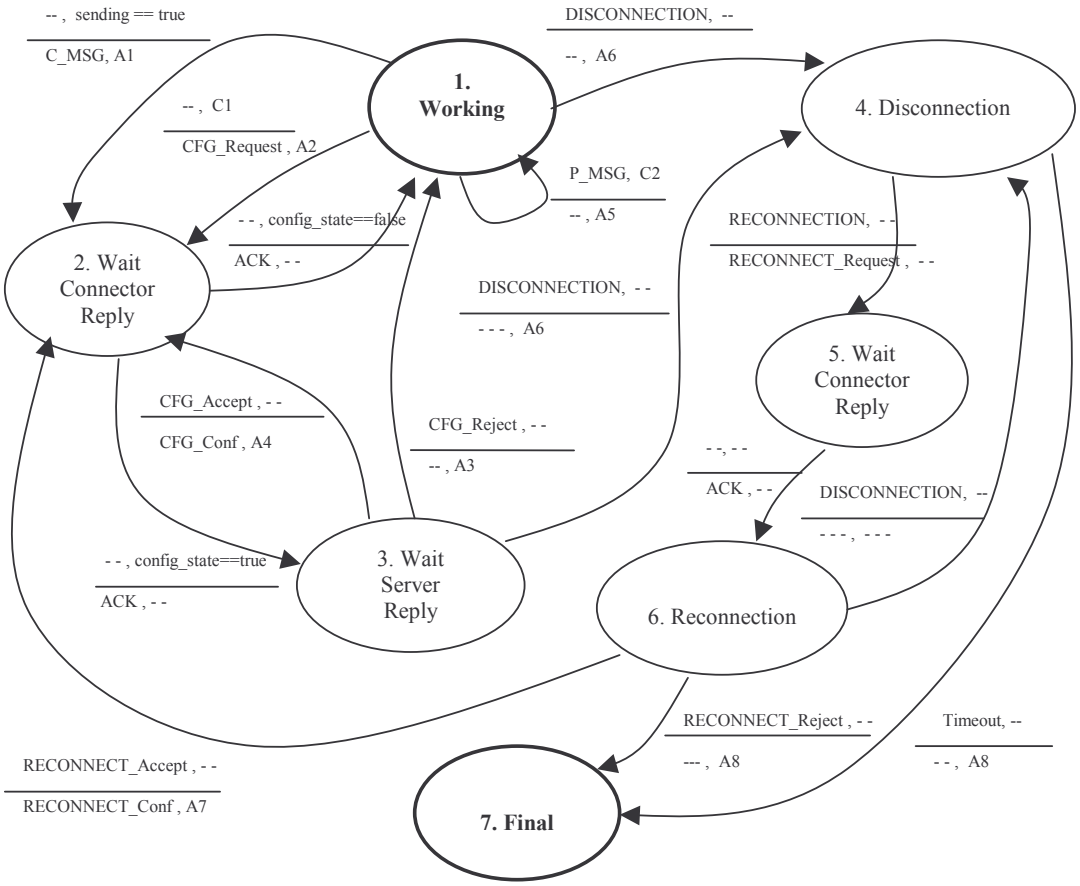


FIG. 7.1 – Machine d'états finis du côté client

fournisseur (CFG_Conf) en reconfigurant le graphe de protocoles et en mettant l'état de l'envoi de message à vrai et de la négociation à faux (action A4). Après l'acquittement de ce message par l'état 2, le client retourne à l'état initial.

Si le client reçoit une notification de déconnexion ($DISCONNECTION$), il arrête l'envoi des messages et ferme le connecteur de transport (action A6). Dans l'état 4, le client se met en attente d'une notification qu'une connexion est de nouveau disponible. A la réception de cette notification, le client passe à l'état 5 en envoyant une demande de reconnexion ($RECONNECT\_Request$). A l'acquittement de cette demande par le connecteur de transport (ACK), le client se met en attente d'une réponse du fournisseur (état 6). Si la réponse est positive ($RECONNECT\_Accept$), il envoie un message de confirmation ($RECONNECT\_Conf$) au fournisseur en réassociant le nouveau connecteur de transport avec le reste du graphe de protocoles (voir section 5.8). L'acquittement de ce message par l'état 2 fait de nouveau passer le client à l'état initial. Par contre, si la réponse est négative, le client passe à l'état final 7 en fermant le graphe de protocoles et la connexion JMS. Si le client ne reçoit pas une notification de reconnexion à l'état 4 avant l'expiration d'un certain délai (*timeout*), il passe également à l'état final 7.

Le client peut également recevoir une notification de déconnexion à l'état 3 (état de négociation). De la même manière que le cas précédent, il passe par les états de reconnexion (4, 5, 6, 2), sauf qu'il retourne à l'état 3 au lieu de l'état initial à cause de la condition $config\_state == true$.

Notons que l'acquittement des messages par les connecteurs de transport (états 2 et 5) empêche le client d'envoyer un deuxième message alors que le premier n'a pas encore été envoyé par le connecteur. Ceci est conforme au comportement de MobileJMS.

FSM du fournisseur

Le FSM du fournisseur est représenté sur la figure 7.2 , où :

- C1 : $C\_MSG.msg\_id == last\_received\_msg\_id + 1$;
- C2 : $CFG\_Request.msg\_id == last\_received\_cfg\_id + 1$;
- C3 : $CFG\_Conf.msg\_id == last\_received\_cfg\_id$;
- C4 : $connected = true$;
- C5 : $connected = false$;
- C6 : $config\_state = true$;
- C7 : $config\_state = false$.

et

- A1 : $last\_received\_msg\_id = C\_MSG.msg\_id$;
- A2 : $last\_msg\_id\_ = \_msg\_id$; msg_id++ ; $P\_MSG.msg\_id = msg\_id$;
- A3 : $last\_received\_cfg\_id = CFG\_Request.msg\_id$; $sending = false$; $config\_state = true$;
- A4 : reconfiguration du graphe en réception ; $CFG\_Accept.msg\_id = CFG\_Request.msg\_id$;
- A5 : $sending = true$; $config\_state = false$;
- A6 : reconfiguration du graphe en envoi ; $sending = true$; $config\_state = false$;
- A7 : $sending = false$; $connected = false$; fermeture du connecteur de transport ;
- A8 : $connected = true$;
- A9 : réassociation avec l'ancien graphe ;
- A10 : fermeture du graphe de protocoles et fermeture de la connexion JMS.

Le FSM du côté fournisseur possède un seul état initial (*working*) et deux états finaux (*working*, *final*). A l'état initial, le fournisseur peut recevoir un message JMS de la part du client (C_MSG), à condition que son identificateur suive celui reçu avec le message précédent (condition C1). Il affecte cet identificateur à l'identificateur du dernier message reçu (action A1). A l'état initial, le fournisseur peut envoyer un message (P_MSG) à condition que l'envoi soit activé ($sending == true$). Il attend ensuite un acquittement (ACK) pour retourner à l'état 1.

Si le fournisseur reçoit une requête de négociation $CFG\_Request$, il passe à l'état 3 à condition que C2 soit vraie. Il arrête en même temps l'envoi de messages vers le client et met l'état de configuration à faux (action A3). S'il décide d'accepter cette requête de changement de configuration, il envoie un message CFG_Accept en reconfigurant le graphe de protocoles en réception.

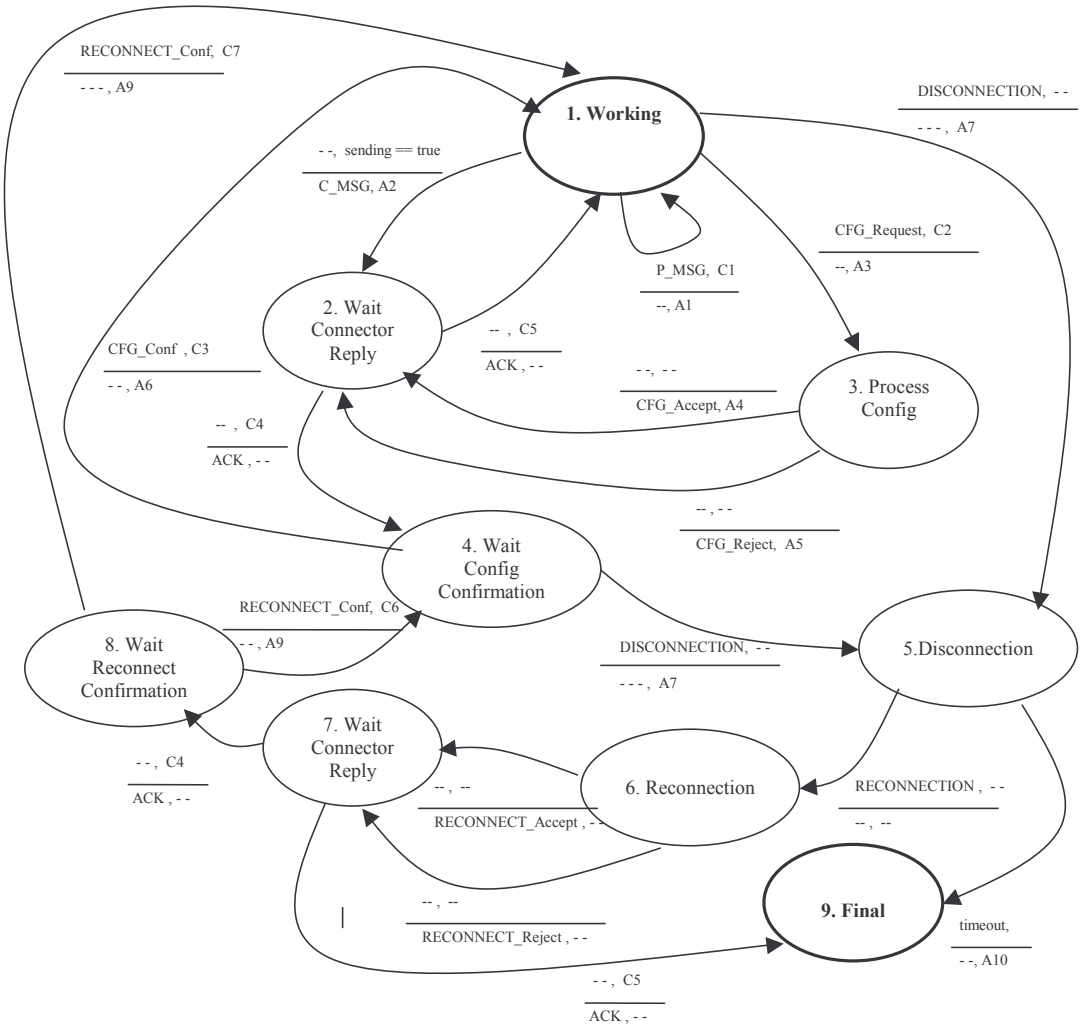


FIG. 7.2 – Machine d'états finis du côté fournisseur

Il attend ensuite l'acquittement du connecteur pour passer à l'état 4. Il attend dans cet état la confirmation du client (CFG_Conf) pour retourner à l'état initial en reconfigurant le graphe en envoi et en réactivant l'envoi de messages (action A6).

Si le fournisseur rejette la requête de négociation, il envoie un message CFG_Reject en réactivant l'envoi de messages et en remettant l'état de configuration à faux (action A5). Il retourne ensuite à l'état initial après la réception de l'acquittement du connecteur.

Dans les états 1 et 4, le fournisseur peut recevoir une notification de la déconnexion du client ($DISCONNECTION$). Il passe dans ce cas à l'état 5 en fermant le connecteur qui communique avec ce client et en arrêtant l'envoi de messages (action A7). Si le fournisseur reçoit une demande de reconnexion du client $RECONNECTION$, il passe dans l'état 6. Il peut alors soit décider d'accepter la reconnexion en envoyant un message $RECONNECT\_Accept$ et en

remettant la valeur de la variable *connected* à vrai, soit la refuser en envoyant au client un message $RECONNECT\_Reject$. Dans le premier cas, il passe à l'état 8 après la réception de l'acquittement du connecteur (ACK). Dans le deuxième cas, il passe à l'état final 9. Après la réception de la confirmation du client ($RECONNEC\_Conf$) dans l'état 8, le fournisseur passe à l'état initial 1 ou à l'état 4 suivant la valeur de la variable $config\_state$ (conditions C6 et C7). Il réassocie également le nouveau connecteur avec l'ancien graphe du client.

Si dans l'état 5, le fournisseur ne reçoit pas de demande de reconnexion avant l'expiration d'un certain délai, il ferme le graphe de protocoles et la connexion JMS du client en passant à l'état final 9.

7.3 Simulation et vérification avec PROMELA/SPIN

7.3.1 Le code PROMELA

Le code PROMELA est construit à partir de la représentation FSM que nous avons décrite précédemment. La simulation de ce code initialise deux processus qui représentent respectivement le client et le fournisseur. Chaque processus instancie un graphe de protocoles composé d'une couche intermédiaire, de la couche d'aiguillage et d'un connecteur. Nous nous sommes contentés d'insérer une seule couche intermédiaire afin de réduire la taille de l'espace d'états. Les objets `Session_High` et `Session_Low` de ces couches sont reliés entre eux par des canaux (*channel*).

Ces deux graphes sont représentés chacun par les processus suivants :

- `Client` représente la couche JMS du client.
- `Provider` représente la couche JMS du fournisseur.
- `Session_High_1` représente l'objet `Session_High` de la couche intermédiaire.
- `Session_Low_1` représente l'objet `Session_Low` de la couche intermédiaire.
- `Session_High_2` représente l'objet `Session_High` de la couche d'aiguillage.
- `Session_Low_2` représente l'objet `Session_Low` de la couche d'aiguillage.

Deux autres processus, `tlayer_clt` et `tlayer_srv` sont initialisés, représentant respectivement les connecteurs du côté client et du côté fournisseur. Ce recours à d'autres processus s'exécutant en parallèle est dû au fait qu'il n'est pas possible de définir des procédures et des 0types de données complexes dans PROMELA. Néanmoins, le comportement est identique à celui de la plate-forme MobileJMS. Un seul message peut transiter à la fois dans le graphe de protocoles grâce à la synchronisation entre l'envoi de message par le client et l'acquittement du connecteur sous-jacent. Cette synchronisation est effectuée à travers un canal (*channel*) qui peut contenir plusieurs messages à la fois, ce qui évite les problèmes d'interblocage entre processus.

Les messages échangés entre le client et le fournisseur sont transmis de manière fiable, sauf dans le cas d'une déconnexion. Dans ce cas, les messages, que le connecteur ne peut plus transmettre à cause de la déconnexion, sont stockés dans une file d'attente. Ils sont transmis après l'achèvement de la procédure de reconnexion.

L'envoi de messages JMS ou de messages de négociation s'effectue de manière aléatoire. Les déconnexions et les reconnexions sont également générées de manière aléatoire par les connecteurs. Les demandes de reconfiguration du graphe de protocoles impliquant une négociation sont également formulées aléatoirement, ainsi que leur acceptation ou leur rejet. L'acceptation entraîne l'insertion ou la suppression de la couche intermédiaire. Le code suivant représente l'implémentation d'un processus `Session_High`. Le mot clé `chan` représente un canal de communication entre plusieurs processus.

```
proctype session_high(byte entity_id, self;chan l2l_self,down) {
  message msg;
  byte type;
  chan l2l_down=[QSZ] of {message};

  l2l_down = down;
  printf("\n%d:Instanciation de la couche Session_High %d\n",entity_id,self);

endSTART:
  do
  :: l2l_self?msg ->
     atomic {
     if
     :: msg.type == rebind ->
       printf("\n%d: restructuration de la couche Session_High %d
                avec%d\n",entity_id,self,msg.bindwlayer);
       l2l_down = msg.bindwlayer
     :: msg.type == terminate ->
        break
     :: else ->
        printf("\n\n%d: Session_High %d:  sending a message to
                %d\n",entity_id,self,l2l_down);
       l2l_down!msg
     fi
     }
  od
}
```

Le processus `Session_High` envoie les messages vers la couche inférieure à travers le canal `l2l_down` et reçoit les messages qui lui sont destinés par le canal `l2l_self`. La boucle principale du processus récupère chaque nouveau message à partir de `l2l_self`. Si le type de message reçu est `rebind` (reconfiguration), le canal `l2l_down` est affecté à la nouvelle couche spécifiée dans le message (`msg.bindwlayer`). Si le message est de type `terminate`, le processus `Session_High` se termine. Si le message est d'un autre type, il est remis directement à la couche inférieure (`l2l_down !msg`).

Le code suivant représente l'implémentation d'un processus `Session_Low`.

```
proctype session_low(byte entity_id; byte self; chan l2l_self, up) {
  chan l2l_up=[QSZ] of {message};
  message msg;
  byte type;

  printf("\n%d: Instanciation de la couche Session_Low
  %d\n",entity_id,self);
  l2l_up = up;

end:
  do
  :: l2l_self?msg ->
     atomic {
     if
     :: msg.type==rebind ->
        printf("\n%d: Restructuration de la couche Session_Low %d
        avec %d\n",entity_id,self,msg.bindwlayer);
       l2l_up = msg.bindwlayer;
     :: msg.type == terminate ->
        break
     :: else ->
        printf("\n\n%d: Session_Low %d: receiving a
        printf("\n\n%d: Session_Low %d: putting the message on
         %d\n",entity_id,self,l2l_up);
        l2l_up!msg
     fi
     }
  od
}
```

Le principe du processus `Session_Low` est le même que celui du `Session_High`, sauf que les messages sont envoyés vers la couche supérieure au lieu de la couche inférieure.

Nous avons introduit dans le code PROMELA des assertions (`assert`) qui permettent de vérifier qu'aucun message ne peut être reçu à un moment inapproprié (par exemple, réception d'une demande de négociation alors qu'une phase de négociation est en cours). Ils permettent également de vérifier si les identificateurs des messages reçus des deux côtés se suivent dans l'ordre.

Nous avons inséré également la proposition temporelle (*temporal claim*) qui est la translation en PROMELA des formules de logique temporelle suivantes :

- `clt_send_msg -> []srv_receive_msg` (si le client envoie un message donné, le fournisseur le reçoit toujours dans un temps fini).
- `clt_send_cfg -> ([](clt_receive_accept || clt_receive_reject))` (si le client envoie une demande de négociation, il reçoit toujours une réponse d'acceptation ou de rejet de la part du fournisseur dans un temps fini).
- `srv_send_accept -> []srv_receive_conf` (si le fournisseur envoie une acceptation à une demande de négociation, il reçoit toujours une confirmation de la part du client dans un temps fini).

La proposition temporelle est la suivante :

```
never { T0_init:
        if
        :: ((((! ((clt_receive_accept)) && ! ((clt_receive_reject)) &&
           (clt_send_cfg)) || ((((! ((srv_receive_conf)) &&
           (srv_send_accept)) || (! ((srv_receive_msg)) &&
           (clt_send_msg)))))) -> goto accept_all
        :: ((clt_send_msg)) -> goto T0_S5
        :: ((clt_send_cfg)) -> goto T0_S10
        :: ((srv_send_accept)) -> goto T0_S15
        fi;
T0_S5:
        if
        :: (! ((srv_receive_msg))) -> goto accept_all
        :: (1) -> goto T0_S5
        fi;
T0_S10:
        if
        :: (! ((clt_receive_accept)) && ! ((clt_receive_reject))) ->
           goto accept_all
        :: (1) -> goto T0_S10
        fi;
T0_S15:
        if
        :: (! ((srv_receive_conf))) -> goto accept_all
        :: (1) -> goto T0_S15
        fi;
accept_all:
        skip
}
```

Cette proposition permet de vérifier que les messages parviennent toujours à leur destinataire dans un temps fini, même s'ils sont entrecoupés par des phases de négociation et de déconnexion/reconnexion. La seule exception à cette règle est le rejet du fournisseur de la demande de reconnexion du client.

7.3.2 La vérification avec SPIN

La vérification est effectuée à l'aide d'un programme C que génère SPIN à partir de la spécification PROMELA. L'exploration exhaustive de tout l'espace d'états est impraticable, car d'un côté, l'aspect aléatoire de la génération d'évènements produit un nombre très important d'états, et de l'autre, le nombre de processus est relativement important. Nous avons donc utilisé la technique du *« Bitstate Hashing »* (voir section 7.1.3). La vérification n'a pas détecté de violation des assertions et des formules LTL (citées ci-dessus), comme le montre la trace d'exécution ci-après.

```
(Spin Version 3.5.3 -- 8 December 2002)
    + Partial Order Reduction

Bit statespace search for:
    never-claim             +
    assertion violations    + (if within scope of claim)
    cycle checks        - (disabled by -DSAFETY)
    invalid endstates   - (disabled by never-claim)

State-vector 1060 byte, depth reached 518020, errors: 0 1.13225e+07
states, stored 6.10606e+07 states, matched 7.23831e+07 transitions
(= stored+matched) 1.83903e+08 atomic steps hash factor: 1.48175
(max size 2^24 states)

Stats on memory usage (in Megabytes): 12047.174   equivalent memory
usage for states (stored*(State-vector + overhead)) 4.194   memory
used for hash-array (-w24) 28.000  memory used for DFS stack
(-m1000000) 38.194  total actual memory usage

unreached in proctype Client
    line 165, state 84, "assert(0)"
    line 259, state 172, "assert(0)"
    (7 of 184 states)
unreached in proctype Server
    line 443, state 141, "assert(0)"
    line 450, state 149, "-end-"
    (2 of 149 states)
unreached in proctype session_high
    line 477, state 18, "-end-"
    (1 of 18 states)
unreached in proctype session_low
    line 505, state 19, "-end-"
    (1 of 19 states)
```

On peut constater que le nombre d'états visité est important (>7 millions) et que la profondeur atteinte dans le graphe est de 518020 (nombre d'états visités à partir de l'état initial). Ces valeurs indiquent que la couverture de l'espace d'état est relativement bonne. SPIN n'as détecté aucune erreur, ce qui signifie qu'aucun état ne viole la proposition de logique temporelle citée ci-dessus. Les seules lignes du code qui n'ont pas été atteintes (`unreached in proctype`) correspondent aux violations d'assertion et à la terminaison des processus, ce qui vérifie la propriété de l'atteignabilité. Nous pouvons donc conclure que les propriétés comportementales, que nous avons retenues, sont valides.

7.4 Synthèse

Nous avons consacré ce chapitre à la validation formelle des mécanismes de négociation et de reconnexion transparente qui forment une partie importante de MobileJMS. Nous avons expliqué le choix de la validation de modèles PROMELA/SPIN par le fait qu'elle fournit des outils appropriés pour l'exploration de l'espace d'état.

Nous avons construit un modèle PROMELA à partir de la représentation abstraite de ces mécanismes avec les machines d'états finis. Ces machines permettent en outre une meilleure compréhension de ces mécanismes. Le modèle PROMELA ne reproduit pas intégralement l'exécution de MobileJMS, comme la gestion des connexions concurrentes des clients. Nous risquons, dans le cas contraire, d'être confronté rapidement au problème de l'explosion combinatoire. Nous envisageons, à ce propos, l'utilisation d'autres outils de vérification de modèles, comme CPN-AMI [53].

Néanmoins, le modèle simule les points critiques dans ces mécanismes comme l'envoi et la réception de messages pendant des phases de négociation ou de reconnexion. Il simule également la reconfiguration du graphe de protocoles en ajoutant ou en supprimant dynamiquement des couches protocolaires.

L'exploration de l'espace d'état effectuée par SPIN a permis de vérifier des assertions et des formules de logique temporelle qui représentent des propriétés comportementales importantes comme l'absence d'interblocage ou la vivacité.

Chapitre 8

Expérimentations et résultats

Nous avons décrit, dans les chapitres précédents, l'architecture de MobileJMS ainsi que son modèle d'adaptation. Ce chapitre étudie les résultats obtenus en utilisant MobileJMS dans diverses conditions d'exécution. Une partie est consacrée à l'évaluation des services de compression et de fragmentation, et des bénéfices qu'ils peuvent apporter en matière de vitesse de transmission des messages.

Les premières expérimentations consistent à définir les configurations logicielles et matérielles (machine, réseau) qui permettent à ces services d'améliorer les performances de MobileJMS. Les mesures observées sont utilisées pour faire une étude comparative par rapport au modèle de prédiction des temps de transmission, que nous avons décrit précédemment. Cette étude nous permet de juger la pertinence du modèle.

La seconde partie de ce chapitre est consacrée à la présentation d'un service témoin que nous avons développé : le service météo. Ce service illustre, à travers l'utilisation de plusieurs types de données, la mise en œuvre des mécanismes d'adaptabilité de MobileJMS face aux variations du contexte d'exécution. En particulier, nous détaillons l'évaluation des bilans dans la procédure du choix des politiques d'adaptation. Les variations du contexte sont simulées afin de pouvoir gérer différents cas de figures.

8.1 Performances des services de compression et de fragmentation

8.1.1 Description de l'environnement de test

L'environnement de test que nous avons adopté est constitué de trois machines : un terminal mobile léger, un PC portable et un PC fixe. Le terminal mobile est un assistant personnel IPAQ H3790 doté d'un processeur Intel StrongArm 206 Mhz. Il possède 32Mo de mémoire vive et 32 Mo de mémoire Flash. Le PC portable est doté d'un processeur Pentium III 933 Mhz et le PC fixe est doté d'un processeur Intel Pentium IV 2.4 Ghz.

Les clients JMS sont déployés sur le terminal mobile qui est exploité par le système Familiar Linux [35]. Ces clients s'exécutent au-dessus de la machine virtuelle Blackdown [8]. Cette machine interprète la version 1.3.1 de la distribution standard de Java (J2SE). Le fournisseur JMS est déployé sur le PC fixe. Le terminal est connecté au fournisseur JMS via l'émulateur Nist Net [71] qui s'exécute sur le PC portable. Nist Net permet au PC portable de se comporter comme un routeur et d'émuler un certain nombre de conditions réseau, comme la bande passante, le délai de bout en bout, le taux de perte de paquets ou le taux de paquets dupliqués. Cet émulateur opère au niveau des paquets IP, il est implémenté comme un module noyau du système Linux.

Le terminal est relié au PC portable avec une connexion 802.11b en mode ad hoc afin d'éviter le trafic concurrent, la bande passante théorique étant de 11 Mb/s. Le PC portable est relié au PC fixe à travers un réseau fixe avec une bande passante de 100 Mb/s (figure 8.1).

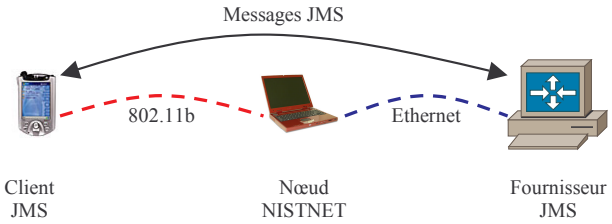


Fig. 8.1 – Emulation des conditions réseau avec Nist Net

8.1.2 Les expérimentations

Les expérimentations consistent à mesurer, pour chaque configuration de MobileJMS, le temps nécessaire aux messages JMS pour être envoyés du client vers le fournisseur JMS et vice-versa. Ces tests sont effectués avec quatre messages représentant les types de données suivants : texte html, fichier exécutable (binaire), une image bmp et un document pdf.

L'objectif de ces expérimentations est d'établir les conditions de bande passante qui permettent à la compression et à la fragmentation d'accélérer la vitesse de transmission.

	Taille initiale (octets)	ZIP 1 taille (octets)	ZIP 4 taille (octets)	ZIP 9 taille (octets)	GZIP taille (octets)
Message html	75629	20373	17860	17123	17019
Message binaire	191554	96165	90222	88187	88274
Message bmp	182868	117145	114033	111652	111652
Message pdf	155171	99870	97858	97357	97406

Tab. 8.1 – Les quatre messages utilisés dans l'expérimentation

Le tableau 8.1 décrit la taille initiale de ces messages ainsi que leur taille après leur compression avec l'algorithme ZIP (avec les taux 1, 4 et 9) et l'algorithme GZIP.

Les messages sont transmis entre le client et le fournisseur en faisant varier la bande passante

réseau. Les mesures sont effectuées au niveau du client mobile. Les valeurs retenues sont la moyenne des temps de transmission de dix messages.

La première expérimentation consiste à mesurer les temps nécessaires au client mobile pour compresser et envoyer les quatre types de messages vers le fournisseur, en utilisant les quatre types de compression. Les figures 8.2 à 8.5 illustrent les résultats pour chaque type de message. La bande passante « normal » correspond à la bande passante fournie par le réseau sans aucune limitation par Nis Net.

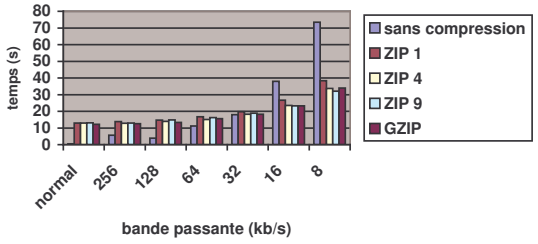


Fig. 8.2 – Temps d'envoi du message html

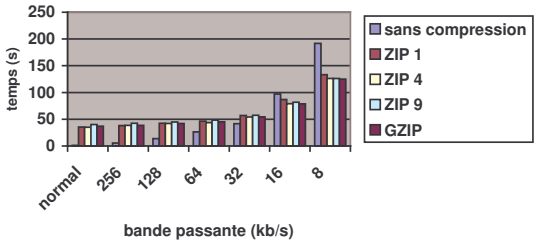


Fig. 8.3 – Temps d'envoi du message binaire

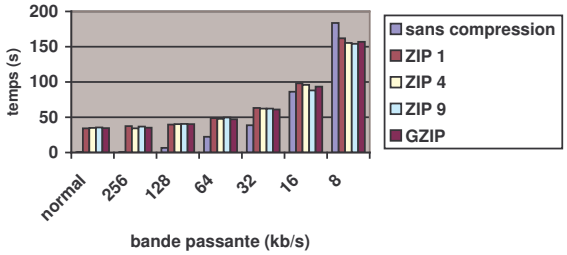


Fig. 8.4 – Temps d'envoi du message bmp

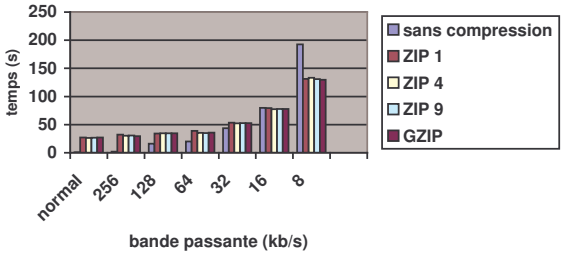


Fig. 8.5 – Temps d'envoi du message pdf

Nous remarquons sur les figures 8.2 à 8.5 que la compression introduit un surcoût important lorsque la bande passante fournie par le réseau n'est pas limitée par Nist Net. Les temps d'envoi de messages compressés sont donc plus importants par rapport à l'envoi de message non compressés. En revanche, lorsque la bande diminue, les temps d'envoi des messages non compressés augmentent plus rapidement que ceux des messages compressés. La compression permet même d'envoyer plus rapidement, les messages html, binaires et pdf en dessous de 16 kb/s. L'avantage de la compression apparaît plus manifestement pour les messages ayant un ratio de compression élevé (html).

Le taux de compression moyen (ZIP 4) apporte un léger avantage par rapport aux autres types de compression. Mais en général, nous n'observons pas de différences significatives entre les quatre types de compression.

La deuxième expérimentation consiste à mesurer les temps nécessaires au client mobile pour recevoir et décompresser les quatre types de messages envoyés par le fournisseur, en utilisant les quatre types de compression. Les figures 8.6 à 8.9 illustrent les résultats pour chaque type de message.

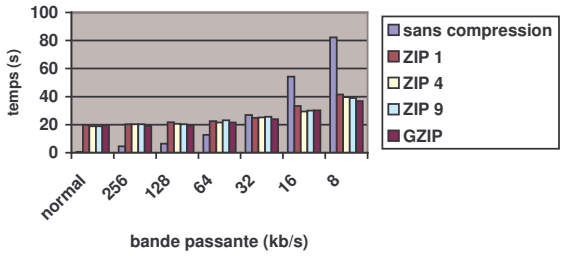


Fig. 8.6 – Temps de réception du message html

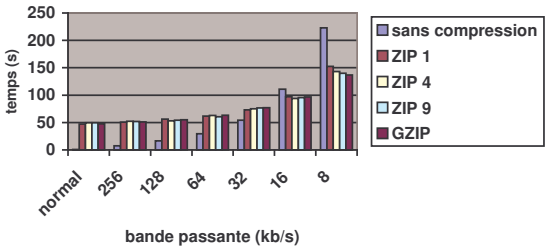


Fig. 8.7 – Temps de réception du message binaire

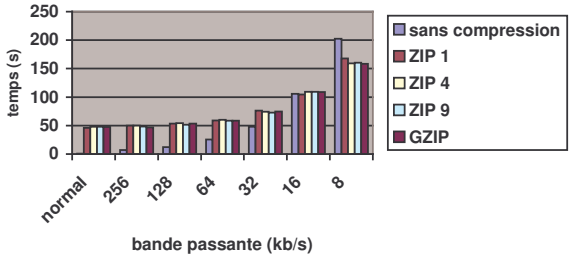


Fig. 8.8 – Temps de réception du message bmp

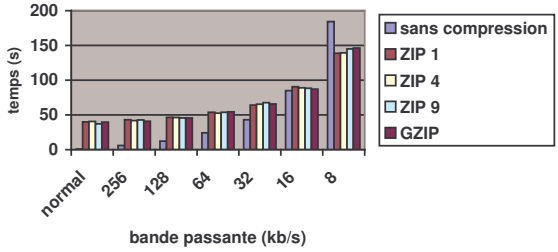


Fig. 8.9 – Temps de réception du message pdf

Les mêmes types de résultats, que ceux de la première expérimentation, peuvent être observés lors de la réception de messages (figures 8.6 à 8.9). Les temps de transmission sont assez proches, ce qui signifie qu'il n'existe pas de grands écarts entre les temps de compression et les temps de décompression.

La troisième expérimentation consiste à mesurer les temps nécessaires au client mobile pour fragmenter, compresser puis envoyer les quatre types de messages vers le fournisseur, en utilisant les quatre types de compression. La taille des fragments est fixée à 1400 octets, ce qui permet à un fragment d'être transporté par un seul paquet. Les résultats sont comparés aux résultats obtenus en n'utilisant que la compression. Les figures 8.10 à 8.13 illustrent les résultats pour chaque type de message.

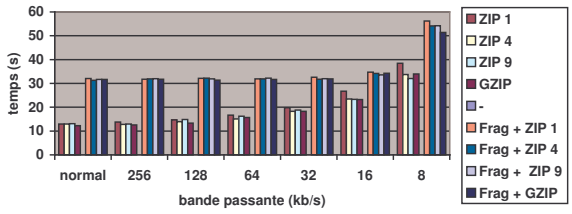


Fig. 8.10 – Temps d'envoi du message html fragmenté

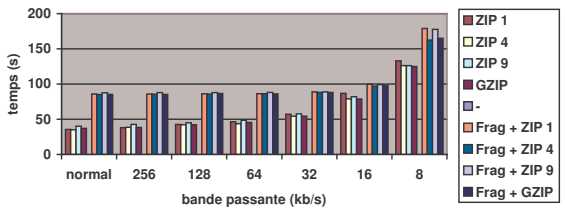


Fig. 8.11 – Temps d'envoi du message binaire fragmenté

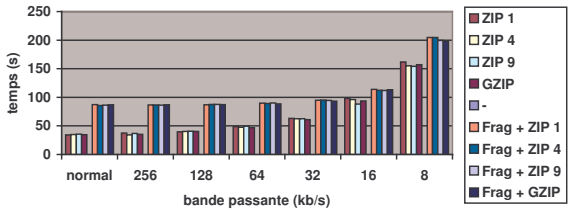


Fig. 8.12 – Temps d'envoi du message bmp fragmenté

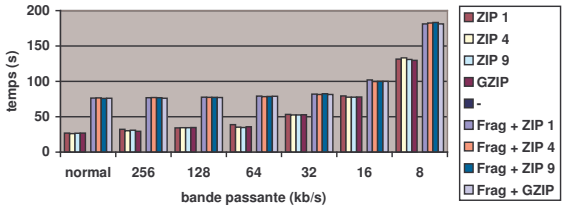


Fig. 8.13 – Temps d'envoi du message pdf fragmenté

La fragmentation introduit un surcoût important, comme nous pouvons l'observer sur les figures 8.10 à 8.13. Les temps d'envoi des messages fragmentés et compressés représentent plus du double des temps d'envoi des messages compressés, lorsque la bande passante est supérieure à 32 kb/s. Mais ces temps restent relativement stables, même avec la diminution de la bande passante. Cela signifie que le temps nécessaire à la génération d'un fragment est largement supérieur à celui nécessaire à son envoi. Le facteur dominant reste donc le temps de fragmentation (voir section 6.4.2).

Nous observons toutefois que les écarts avec les autres configurations de MobileJMS diminuent avec la diminution de la bande passante. Nous pouvons donc penser que la fragmentation pourrait produire de meilleurs résultats si elle est implémentée de manière plus efficace. En particulier, la génération d'un fragment, qui repose sur la gestion de mémoire de Jonathan, nécessite la copie des tableaux de données en mémoire, qui est assez coûteuse du point de vue du temps d'exécution.

La quatrième expérimentation consiste à mesurer les temps nécessaires pour recevoir, décompresser puis défragmenter les quatre types de messages envoyés par le fournisseur, en utilisant les quatre types de compression. Les figures 8.14 à 8.17 illustrent les résultats pour chaque type de message.

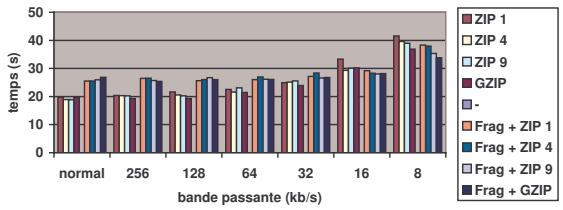


Fig. 8.14 – Temps de réception du message html fragmenté

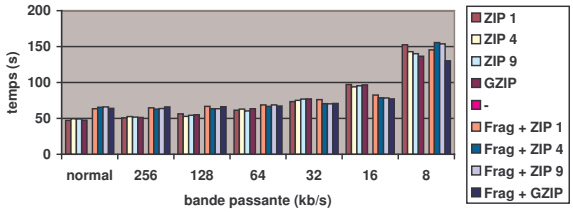


Fig. 8.15 – Temps de réception du message binaire fragmenté

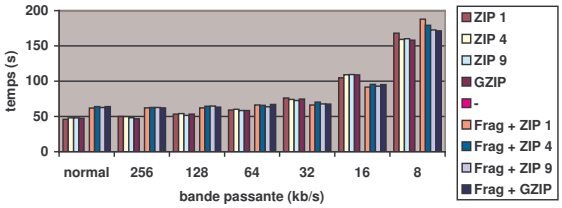


Fig. 8.16 – Temps de réception du message bmp fragmenté

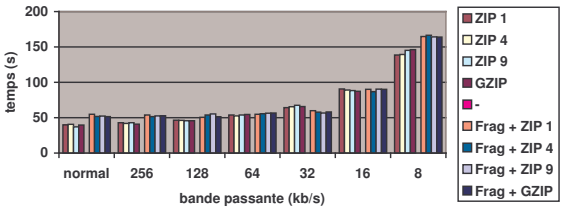


Fig. 8.17 – Temps de réception du message pdf fragmenté

La fragmentation est généralement plus efficace lors de la réception des messages (figures 8.14 à 8.17). Lorsque la bande passante est inférieure à 32 kb/s, nous pouvons observer un avantage par rapport à la compression pour les messages html et binaire. Cela est principalement dû au fait que la défragmentation des messages est moins coûteuse que la fragmentation (moins de copie de tableaux).

Discussion

D'une manière générale, nous pouvons conclure que les services de compression et de fragmentation permettent de réduire les temps de transmission dans certaines situations. La fragmentation, associée à la compression, peut même s'avérer un moyen très efficace pour la réception des messages de tailles importantes et qui possèdent un bon ratio de compression. Ce cas de figure correspond aux applications typiquement utilisées dans les environnements mobiles. Les performances de ces services peuvent même être améliorées en optimisant leurs implémentations. Il faut noter également qu'une bonne partie de ces performances dépend de la machine virtuelle Java. Nous pensons donc que les performances peuvent être sensiblement améliorées quand des JVMs plus rapides seront disponibles.

8.2 Evaluation du modèle de prédiction des temps de transmission

A partir des résultats obtenus dans la section précédente, nous pouvons établir la marge d'erreur du modèle de prédiction de transmission, que nous avons présenté dans la section 6.4.1. Cette marge est calculée à partir des différences entre les valeurs mesurées et les valeurs prédites dans les mêmes conditions de bande passante.

8.2.1 Comparaison des résultats

Nous devons, tout d'abord, évaluer les paramètres suivants : ratio de compression, vitesse de compression *cs* et vitesse de décompression *ds* correspondant à chaque type de message. En compressant un échantillon composé de plusieurs messages de chaque type, nous avons obtenu les

moyennes et les écarts-types des ratios de compression qui sont présentés dans le tableau 8.2. Par ailleurs, nous avons évalué les moyennes de *cs* et *ds* correspondant à ces types de messages, les résultats sont présentés dans le tableau 8.3 et 8.4. Les écarts-types des vitesses de compression et de décompression ne sont pas très élevés. Ces valeurs sont donc assez représentatives pour chaque type de message. En revanche, ceux des ratios sont plus importants, car les ratios dépendent du type mais surtout du contenu du message.

	ZIP 1		ZIP 4		ZIP 9		GZIP	
	moyenne	écart-type	moyenne	écart-type	moyenne	écart-type	moyenne	écart-type
html	3,90	0,659	4,39	0,957	4,51	0,897	4,55	0,752
binaire	1,87	0,964	2,05	1,189	2,22	0,645	2,21	1,165
bmp	1,80	1,06	1,99	1,77	2,32	0,777	2,39	0,847
pdf	1,60	0,929	1,67	1,205	1,75	1,105	1,74	0,905

Tab. 8.2 – Ratios de compression suivant le type de message

	ZIP 1		ZIP 4		ZIP 9		GZIP	
	moyenne (ko/s)	écart-type	moyenne (ko/s)	écart-type	moyenne (ko/s)	écart-type	moyenne (ko/s)	écart-type
html	5,718	0,159	5,639	0,152	5,629	0,228	5,635	0,317
binaire	5,400	0,164	5,314	0,168	4,802	0,133	5,193	0,156
bmp	5,341	0,216	5,251	0,177	5,125	0,120	5,190	0,162
pdf	5,328	0,132	5,274	0,205	5,380	0,155	5,366	0,125

Tab. 8.3 – Vitesses de compression suivant le type de message

	ZIP 1		ZIP 4		ZIP 9		GZIP	
	moyenne (ko/s)	écart-type	moyenne (ko/s)	écart-type	moyenne (ko/s)	écart-type	moyenne (ko/s)	écart-type
html	3,873	0,179	3,834	0,212	3,834	0,178	3,871	0,236
binaire	3,851	0,111	3,856	0,112	3,987	0,198	3,982	0,106
bmp	4,013	0,245	4,018	0,156	3,954	0,102	3,882	0,137
pdf	3,963	0,116	3,876	0,180	3,832	0,099	3,913	0,190

Tab. 8.4 – Vitesses de décompression suivant le type de message

Par ailleurs, nous avons estimé les temps nécessaires à la fragmentation et à la défragmentation. La génération d'un fragment de 1400 octets nécessite en moyenne 1,93 s.

Comme exemple, prenons le cas de l'envoi du message html (75629 octets) lorsque la bande passante est de 64 kb/s. Les valeurs obtenues avec le modèle de prédiction sont les suivantes :

- **Envoi du message sans compression :** la formule qui s'applique dans ce cas est (6.1) (section 6.4.1) :

$$t_1 = \frac{s}{bw} = \frac{75629}{8192} = 9,23\ s$$

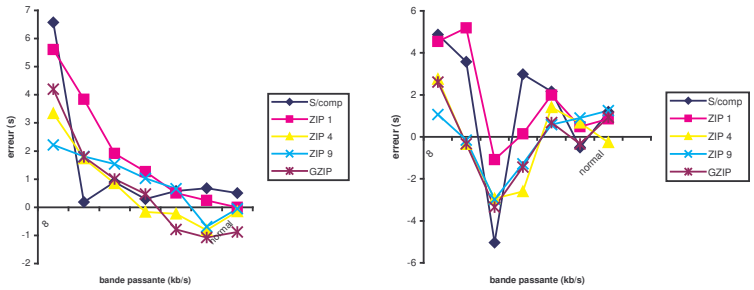
- **Envoi du message compressé avec GZIP :** la formule qui s'applique dans ce cas est (6.2)

$$t_2 = \frac{s}{cs} + \frac{s}{ratio \times bw} = \frac{75629}{5,635 \times 1024} + \frac{75629}{4,55 \times 8192} = 15,14\ s$$

– **Envoi du message fragmenté et compressé avec GZIP :** Le nombre de fragments nécessaires pour ce message est 13. En sachant que le temps de compression d'un fragment est de 0,24 s et que le temps d'envoi d'un fragment sur le réseau est de 0,04 s : le temps de traitement local est plus important que celui de transmission sur le réseau. La formule qui s'applique donc est (6.8)(section 6.4.1) :

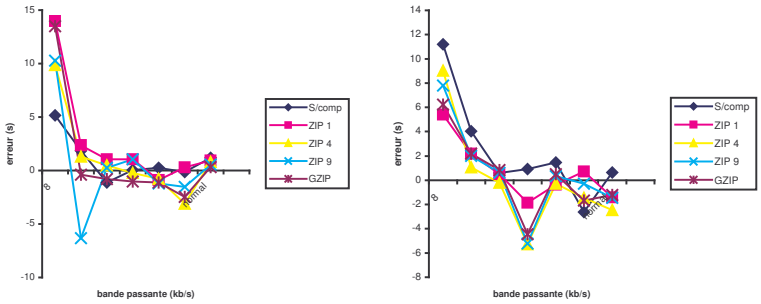
$$t_3 = n \times t_{frag} + n \times t_{comp} + t_s = 13 \times 1,93 + 13 \times 0,24 + 0,04 = 28,25\ s$$

Les figures 8.18 et 8.19 représentent les différences, entre les temps mesurés et les temps prédits, lors de l'envoi de chaque type de message en utilisant la compression. Les figures 8.20 et 8.21 représentent ces différences lors de la réception de messages. Les figures de 8.22 à 8.25 représentent ces différences lors de l'utilisation de la fragmentation.



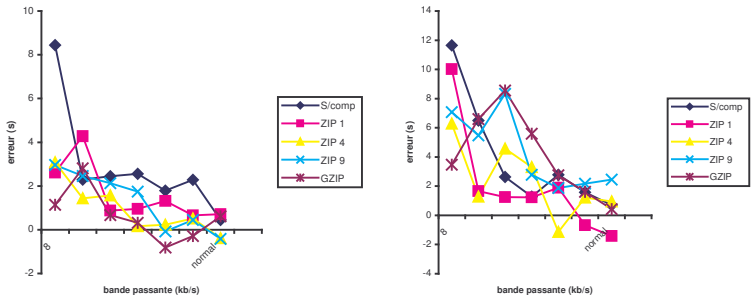
(a) envoi du message html *(b) envoi du message binaire*

Fig. 8.18 – Écarts des temps d'envoi des messages compressés (1)



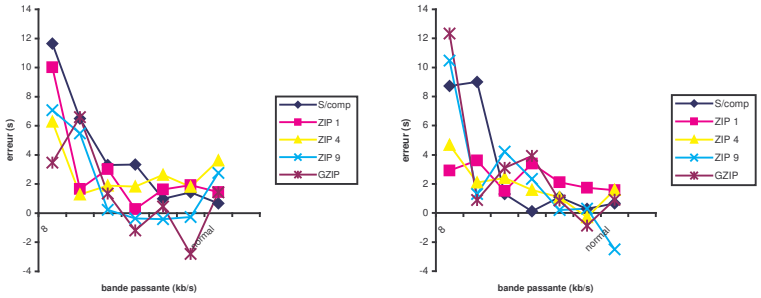
(c) envoi du message bmp *(d) envoi du message pdf*

Fig. 8.19 – Écarts des temps d'envoi des messages compressés (2)



(a) réception du message html *(b) réception du message binaire*

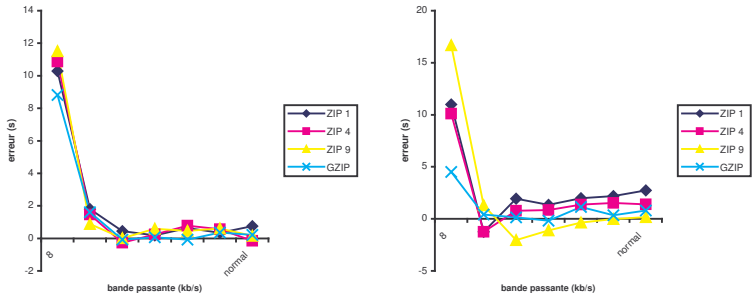
Fig. 8.20 – Écarts des temps de réception des messages compressés (1)



(c) réception du message bmp *(d) réception du message pdf*

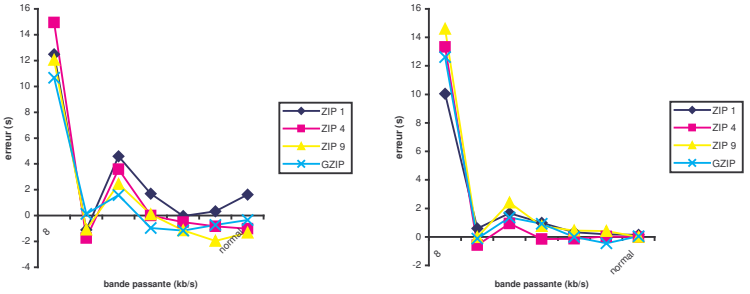
Fig. 8.21 – Écarts des temps de réception des messages compressés (2)

Les résultats montrent, en général, que les différences entre les valeurs mesurées et les valeurs prédites ne sont pas très importantes. Ces différences oscillent entre 1 et 8 s sauf lorsque la bande passante est de 8 kb/s. Ces erreurs représentent donc, au maximum, 15 à 20% du temps mesuré. Ce résultat permet d'obtenir une marge sure entre les valeurs obtenues pour chaque niveau de bande passante. Nous ne devons pas non plus négliger le fait que l'exécution se déroule sur la machine virtuelle Java qui n'offre aucune garantie sur les délais d'exécution. Elle peut donc introduire des distorsions. Les taux d'erreurs, particulièrement élevés, obtenus à 8 kb/s, s'expliquent par le fait que Nist Net attribue moins de bande passante que celle qui doit être attribuée théoriquement (zone d'erreur).



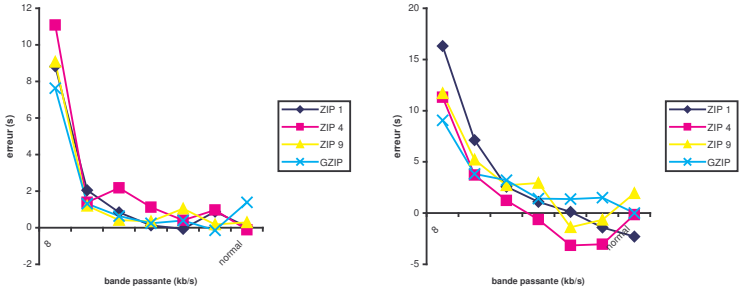
(a) envoi du message html *(b) envoi du message binaire*

Fig. 8.22 – Écarts des temps d'envoi des messages compressés et fragmentés (1)



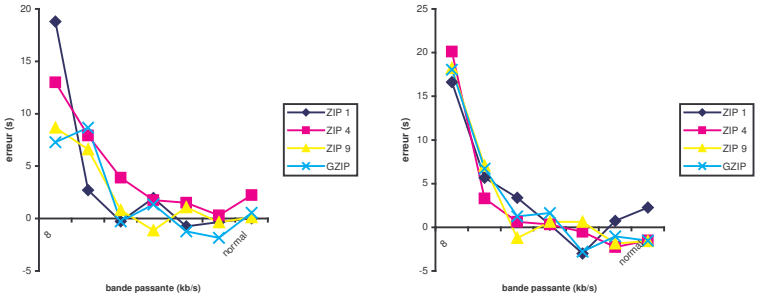
(c) envoi du message bmp *(d) envoi du message pdf*

Fig. 8.23 – Écarts des temps d'envoi des messages compressés et fragmentés (2)



(a) réception du message html *(b) réception du message binaire*

Fig. 8.24 – Écarts des temps de réception des messages compressés et fragmentés (1)



(c) réception du message bmp *(d) réception du message pdf*

Fig. 8.25 – Écarts des temps de réception des messages compressés et fragmentés (1)

Le cas des messages fragmentés est particulièrement intéressant, car le modèle apparaît comme plus précis. Nous pouvons l'expliquer par le fait qu'il existe moins de risque d'erreurs lorsque le calcul s'effectue sur des fragments de petites tailles.

En conséquence, nous pouvons affirmer, que le modèle, bien qu'il ne soit pas toujours précis, fournit néanmoins une bonne approximation concernant les temps de transmission. Cette approximation permet, par exemple, de distinguer dans quel cas la compression améliore les performances, et dans quel cas, au contraire, elle les détériore. Les marges d'erreurs du modèle ne modifient pas l'ordre des performances observées dans la réalité entre les configurations suivantes : configuration de base, compression, fragmentation et compression.

Par ailleurs, il y a très peu de différences de performance entre les quatre types de compression. Le peu d'écart, qui existe entre les ratios et les vitesses de compression de ces quatre types, se reflète dans les valeurs mesurées. Il nous paraît donc plus approprié de ne retenir qu'un seul type de compression, ce qui permet de réduire le traitement nécessaire lors du choix des politiques.

8.3 Un service témoin : les données météo

Afin d'illustrer les différentes possibilités de MobileJMS et du modèle d'adaptation, nous avons conçu un service témoin. Ce service permet à un utilisateur mobile de recevoir périodiquement des données météorologiques. Ces données sont fournies par les deux serveurs *meteo_fr* et *meteo_de* qui correspondent respectivement à la météo en France et en Allemagne. Dans la terminologie JMS, ces serveurs sont des producteurs de message alors que le service client est un consommateur de messages. Les différents types de données fournies par ces serveurs sont des images haute résolution, des images basse résolution et du texte. L'architecture de l'application est représentée par la figure 8.26.

Un ensemble de politiques d'adaptation est associé à cette application. Parmi les paramètres définis dans ces politiques :

– *HighBW* : BW>30 kb/s (bande passante), la localisation est inconnue, le serveur est *me-*

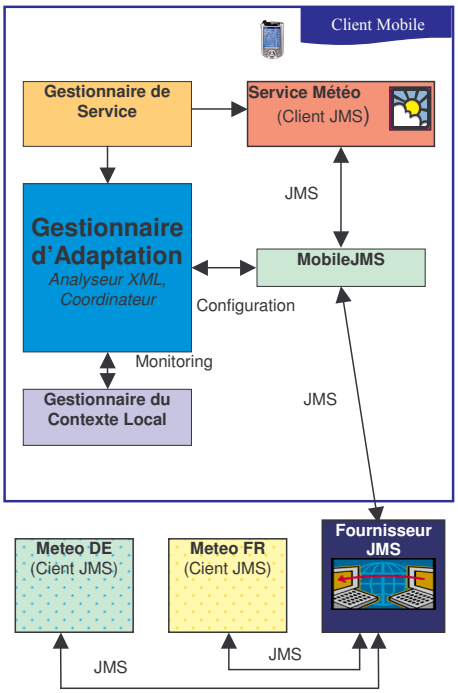


Fig. 8.26 – Architecture du service météo

teo_fr et le type de données est « *LargeImage* ».
- *MediumBW* : BW>10 kb/s, la localisation est inconnue, le serveur est *meteo_fr* et le type de données est « *SmallImage* ».
- *LowBW* : BW>3 kb/s, la localisation est inconnue, le serveur est *meteo_fr* et le type de données est « *Text* ».
- *HighBW_France* : BW>30 kb/s, la localisation est en France, le serveur est *meteo_fr* et le type de données est « *LargeImage* ».
- *MediumBW_France* : BW>10 kb/s, la localisation est en France, le serveur est *meteo_fr* et le type de données est « *SmallImage* ».
- *LowBW_France* : BW>3 kb/s, la localisation est en France, le serveur est *meteo_fr* et le type de données est « *Text* ».
- *HighBW_Germany* : BW>30 kb/s, la localisation est en Allemagne, le serveur est *meteo_de* et le type de données est « *LargeImage* ».
- *MediumBW_Germany* : BW>10 kb/s, la localisation est en Allemagne, le serveur est *meteo_de* et le type de données est « *SmallImage* ».

Le mode d'exécution utilisé est le mode de qualité de service. Le calcul des bilans s'effectue donc par l'algorithme défini dans la section 6.5.1. Pour les besoins de simplification, nous n'avons retenu dans cette section que la configuration de base et la configuration « compression GZIP » de MobileJMS. Les vitesses de transmission prennent uniquement en compte les vitesses de réception, car les envois de message ne constituent, dans ce service, qu'une très faible partie du trafic.

Par ailleurs, les paramètres du contexte d'exécution peuvent être modifiés à l'aide du simu-

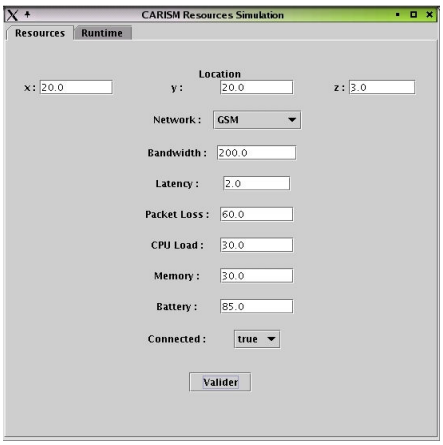


Fig. 8.27 – Le simulateur de ressources

lateur de ressources (figure 8.27). Suivant les valeurs de ces paramètres, plusieurs scénarios sont possibles. Parmi lesquels :

Bande passante élevée

La bande passante du contexte est de 40 kb/s et la localisation est en France. Si nous appliquons la politique *HighBW_France*, en sachant que le type des messages est bmp, nous obtenons les résultats suivants :

- *Sans compression* : la vitesse de transmission $v(s\_comp)$, dans ce cas, est égale à la bande passante du contexte (40 kb/s). Elle est supérieure à la bande passante minimale qui est requise par la politique *HighBW_France* ($P.bw > 30kb/s$). Cette politique est donc applicable avec cette configuration et le bilan est :

$$Bilan = P.bw \times v(s\_comp) = 30 \times 40 = 1200$$

- *Compression GZIP* : avec un ratio de compression de 3,24 et une vitesse de décompression de 20 kb/s, la vitesse de transmission est calculée de la manière suivante :
  Pour la réception d'un message de 10 ko, le temps nécessaire est :

$$t_{recep} = \frac{s}{cs} + \frac{s}{ration \times bw} = \frac{10}{20} + \frac{10}{3,24 \times 40} = 0,577s$$

  La vitesse est donc $v(comp\_gzip)$ =17,33 kb/s.
  Cette vitesse est inférieure à la bande passante requise dans la politique (30 kb/s). Cette configuration n'est donc pas applicable avec cette politique, le bilan est 0.
  Si nous appliquons la politique *MediumBW_France* (10 kb/s), le bilan avec la configuration « sans compression »est 400.

Le meilleur bilan est donc fourni par *HighBW_France* (1200) en utilisant la configuration MobileJMS « sans compression ». Cette politique est donc choisie. L'utilisateur reçoit dans ce cas du serveur *meteo_fr* des images haute résolution et non compressées concernant la météo en France (figure 8.28).

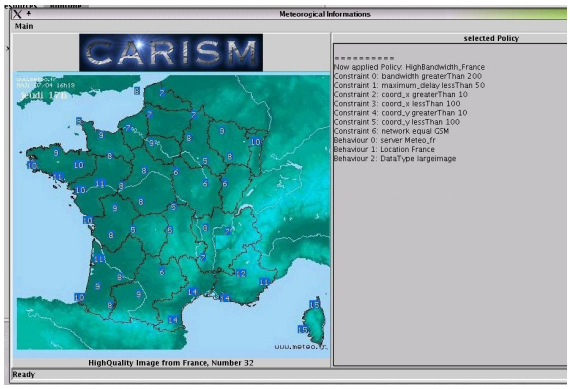


Fig. 8.28 – Réception d'images haute résolution de la météo en France

Diminution de la bande passante

La bande passante diminue à 13 kb/s et la localisation reste en France. La politique *HighBW_France* ne peut pas être appliquée dans ce cas, car avec la configuration « sans compression », la vitesse de transmission (13 kb/s) est inférieure à la bande passante de la politique (30 kb/s). La configuration « compression GZIP », qui ne fournit qu'une vitesse de transmission de de 13,5 kb/s, n'est également pas applicable.

Ces vitesses de transmission sont, en revanche, suffisantes pour l'application de la politique *MediumBW_France* ($P.bw$ >10 kb/s). Les bilans sont :

- *sans compression* : $Bilan(s\_comp) = P.bw \times v(s\_comp) = 10 \times 13 = 130$;
- *compression GZIP* : $Bilan(comp\_gzip) = P.bw \times v(comp\_gzip) = 10 \times 13,5 = 135$.

La meilleure configuration dans ce cas est « compression GZIP ». L'utilisateur reçoit dans ce cas du serveur *meteo_fr* des images basse résolution concernant la météo en France (figure 8.29).

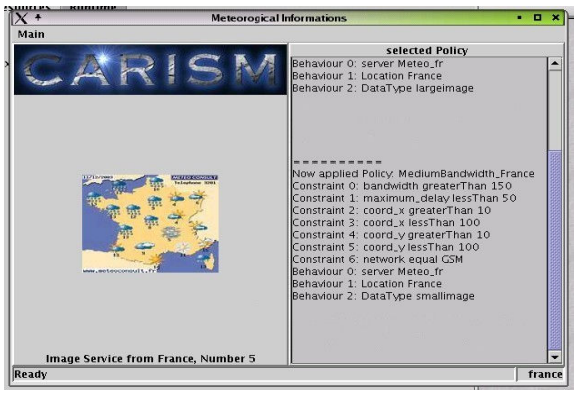


Fig. 8.29 – Réception d'images basse résolution de la météo en France

Faible bande passante

La bande passante du contexte est de 3,5 kb/s et la localisation est à Paris. La politique *MediumBW_France* (>10kb/s) ne peut plus s'appliquer, même en utilisant la compression GZIP, car la vitesse de transmission n'est que de 9,65 kb/s.

La politique qui s'applique dans ce cas est *LowBW_France* (>3 kb/s). La configuration « compression GZIP » est utilisée car elle fournit un bilan (28,95) plus important que celle de la configuration « sans compression » (10,5), grâce à une meilleure vitesse de transmission. Par conséquent, l'utilisateur reçoit de *meteo_fr* des informations textuelles concernant la météo à Paris qui correspondent à celles de la France (figure 8.30)

FIG. 8.30 – Réception d'informations textuelles de la météo à Paris

8.4 Synthèse

Nous avons présenté dans ce chapitre les résultats concernant l'exécution de MobileJMS sur terminal mobile. Nous avons en particulier mesuré le coût des services de compression et de fragmentation dans des conditions différentes de bande passante. Nous avons ainsi pu établir dans quelles conditions ils permettent d'améliorer la vitesse de transmission des messages.

Nous avons, par ailleurs, évalué le modèle de prédiction présenté dans le chapitre 6. Les résultats obtenus montrent qu'il permet de fournir, pour toutes les configurations utilisées de MobileJMS, une bonne estimation des temps d'envoi et de réception des messages. Les marges d'erreur observées sont généralement inférieures aux écarts qui existent entre les résultats obtenus avec chaque configuration de MobileJMS. Cette constatation permet d'affirmer que la meilleure configuration de MobileJMS, déduite à partir du modèle de prédiction, correspond dans la majorité des cas à la meilleure configuration de MobileJMS observée dans la réalité. Cette équivalence représente l'objectif essentiel du modèle de prédiction.

Enfin, nous avons développé un service témoin qui permet d'illustrer les décisions du modèle d'adaptation. Ce service est associé à un ensemble de politiques d'adaptation qui répondent aux différents paramètres de bande passante et de localisation qui peuvent être rencontrés. En

particulier, ces politiques permettent au service de recevoir les données suivant des formats différents (images ou texte). Nous avons décrit l'évaluation des bilans, dans le mode de qualité de service, par rapport aux besoins de chaque politique et aux différents états du contexte d'exécution.

Chapitre 9

Conclusion générale

9.1 Intergiciels mobiles

Grâce à la convergence des technologies des télécommunications et de l'informatique, l'usage des services mobiles ne se réduit plus aux communications téléphoniques, mais à une vision plus large du futur numérique. Le monde devient plus dépendant de ces services, mais la capacité des infrastructures actuelles à répondre au besoin croissant des utilisateurs demeure une question ouverte. L'un des objectifs principaux est de rendre universels et accessibles, à n'importe quel moment à n'importe quel endroit, des services, comme l'accès au Web et aux systèmes d'information des entreprises, ou l'interaction avec les ressources disponibles dans le voisinage physique.

Dans ce contexte, les intergiciels peuvent apporter une forte valeur ajoutée. Les intergiciels mobiles actuels se caractérisent par une grande diversité du point de vue de leur architecture et du type d'applications et d'environnements ciblés. La variabilité des environnements mobiles, entre la nomadicité et l'omniprésence, et la variabilité des architectures matérielles génèrent des paramètres qui sont difficiles à isoler dans une seule solution à usage général. Mais à défaut de mettre au point ce modèle universel, les recherches ont permis d'établir un ensemble de concepts bien définis comme l'adaptabilité, la sensibilité au contexte, l'interaction asynchrone, la configurabilité ou l'interopérabilité. Ces concepts préfigurent ce que sera l'intergiciel mobile de demain.

Les intergiciels orientés messages MOMs constituent l'une des voies de recherche prometteuses. Comme l'atteste le nombre de produits disponibles actuellement sur le marché (iBus//mobile [92], MSMQ [65], IBM EveryPlace [45]), les MOMs mobiles proposent une solution intéressante aux problèmes de la nomadicité : mobilité (*roaming*), connexion intermittente, etc. Cependant, le fonctionnement de ces intergiciels reste sur plusieurs aspects semblable à ceux des MOMs fixes. Ils sont également moins adaptés aux environnements diffus où il n'existe pas d'accès à une infrastructure fixe. Le souci d'interopérabilité et de compatibilité est sûrement l'une des raisons principales. En outre, les MOMs mobiles doivent également évoluer pour prendre en compte d'autres considérations comme l'adaptabilité et la sensibilité au contexte.

Par ailleurs, la multiplication des standards de réseaux sans fil et leurs caractéristiques différentes a fait surgir le paradigme de l'accès multi-réseaux. Comme il est improbable qu'une seule technologie sans fil puisse répondre à tous les besoins, l'un des services importants que peuvent assurer les intergiciels mobiles est de permettre aux applications de communiquer de manière transparente à travers plusieurs réseaux.

9.2 Réalisations

Nous avons conçu et réalisé, MobileJMS, un intergiciel basé sur la spécification JMS. MobileJMS intègre dans son module de communication des services qui permettent de gérer la transmission de messages dans différents contextes. En particulier, la compression et la fragmentation permettent d'optimiser la transmission si le réseau se caractérise par une bande passante limitée et des pertes de paquets importantes.

Le service de stockage et ré-émission propose une solution à l'un des problèmes importants de la spécification JMS qui est la forte dépendance entre les applications et la connexion réseau. La possibilité d'émettre des messages même en étant déconnecté, est un facteur important pour la performance, la maîtrise des coûts et la convivialité des applications mobiles. Ce service met en œuvre une gestion qui favorise la transmission de messages importants lorsque les connexions sont brèves et de faible qualité.

MobileJMS permet de modifier et de reconfigurer, en cours d'exécution, les services (les couches protocolaires) utilisées par le module de communication. Ces modifications donnent lieu à une négociation entre le client et le fournisseur JMS. Cette négociation est nécessaire pour, d'une part, pourvoir interpréter correctement les messages, et d'autre part, vérifier la disposition de l'autre entité à utiliser la nouvelle configuration proposée.

La structure de module de communication de MobileJMS (graphe de protocoles) permet de répondre à deux problématiques des environnements mobiles : l'accès multi-réseaux et la connexion intermittente. Le graphe de protocoles permet d'associer une seule connexion logique JMS à plusieurs connexions de bas niveau. Cette multiplicité concerne aussi bien les réseaux utilisés que les protocoles de transport. Nous avons notamment introduit un mécanisme qui permet au fournisseur de pouvoir identifier un client utilisant plusieurs connexions au même moment.

Les déconnexions brèves et imprévues sont interceptées et occultées aux applications par le graphe de protocoles. Après la reconnexion réseau, le client déconnecté déclenche une procédure de reconnexion JMS transparente qui lui permet de reprendre le déroulement normal de ses activités. Les développeurs d'applications JMS n'ont pas ainsi besoin de se préoccuper de ces évènements.

Les mécanismes de négociation et de reconnexion ont fait l'objet d'une vérification formelle à l'aide de la vérification de modèles PROMELA/SPIN. Cette vérification a permis de valider des propriétés importantes comme la vivacité ou l'absence d'interblocage et de comportements non spécifiés .

Les décisions relatives à l'adaptation de MobileJMS face aux variations du contexte sont prises dans un cadre plus général que celui du niveau intergiciel. Nous avons conçu et mis en œuvre un modèle qui fait intervenir les préférences des utilisateurs, les besoins des applications, le contexte d'exécution et la configuration de MobileJMS. Les besoins des applications sont exprimés sous forme de politiques d'adaptation. Une politique définit l'état du contexte d'exécution qui permet à une application de satisfaire des besoins donnés.

Le choix de la politique la plus adaptée au contexte actuel s'effectue par rapport au type d'usage que l'utilisateur souhaite faire de l'application. Nous avons retenu les critères de vitesse de transmission et d'économie d'énergie. Nous avons défini un modèle qui relie ces critères aux différentes configurations de MobileJMS. En particulier, nous avons étudié la relation entre la compression et la fragmentation d'un côté, et ces critères de performance de l'autre. Cette relation ainsi que la relation entre les besoins des applications et les contraintes du contexte font l'objet d'un modèle qui détermine la configuration MobileJMS la plus adaptée. Cette configuration est déduite par rapport à deux modes d'exécution : le mode de qualité de service et le mode d'économie d'énergie.

Les performances des services de compression et de fragmentation ont été mesurées. Les résultats ont permis de constater le gain de performance apportés par ces services dans certaines conditions réseau. Nous avons, par ailleurs, établit la pertinence du modèle de prédiction. L'écart entre les valeurs mesurées et les valeurs déterminées, par prédiction des temps de transmission des messages, a ainsi été évalué.

Enfin, nous avons développé un service témoin (service météo) basé sur MobileJMS illustrant les décisions d'adaptation dans différents contextes d'exécution.

9.3 Perspectives

La réalisation et les tests effectués avec MobileJMS nous ont permis de dégager des pistes de recherche à approfondir. En particulier, le modèle d'adaptation devra tenir compte du partage des ressources du terminal par plusieurs services.

Optimisation des performances La vitesse d'exécution de MobileJMS sur les assistants personnels IPAQ est beaucoup moins importante que celle sur les machines fixes. L'une des raisons principales est la performance de la machine virtuelle Java. L'occupation de la mémoire est relativement élevée par rapport aux capacités des terminaux actuels. L'un des facteurs pesants est également le temps nécessaire à l'instanciation des classes. La machine virtuelle J2ME [97] peut constituer une alternative, mais nous pensons que la distribution standard J2SE reste un bon choix, car elle offre des outils plus riches pour le développement d'applications mobiles, comme les structures de données complexes. L'évolution inéluctable des capacités des terminaux mobiles ainsi que celle des performances des machines virtuelles permettront sans doute d'atteindre de meilleures performances dans un futur proche.

Par ailleurs, nous avons entrepris l'optimisation de MobileJMS. L'introduction d'autres patrons de conception permettra, par exemple, de réutiliser les objets déjà présents et d'éviter ainsi

la surcharge due à l'instanciation. Nous avons pu constater une utilisation excessive des exécutions concurrentes (*multi-threading*) qui peut être évitée. Les copies des tableaux en mémoire peuvent également être minimisées.

Généralisation des services à valeur ajoutée Les services à valeur ajoutée, que nous avons développé, sont de nature générique. Des services plus spécifiques au type de données peuvent être introduits à MobileJMS. La connaissance sémantique des données peut réduire de manière plus efficace leur taille et les proposer selon des qualités progressives. Des mécanismes de raffinement, dégradation et distillation (section 2.3.2) peuvent être intégrés pour améliorer, notamment, la transmission des données multimédia.

De manière générale, les services à valeur ajoutée constituent un concept large qui peut englober tout type de traitement qui ne dépend pas de la logique JMS. L'envoi groupé de messages et l'encryptage sont des exemples de services qui peuvent être intégrés.

Composition des politiques d'adaptation La relation entre les politiques d'adaptation est basée sur la spécialisation. Chaque politique est entièrement indépendante de l'autre et correspond à un seul état du contexte. Nous pouvons envisager d'autres types de relations, comme la composition. Dans le cas où il n'existerait pas de politique qui puisse s'appliquer au contexte courant, le modèle devra appliquer une politique qui peut être la composition d'autres politiques. Nous pouvons imaginer, par exemple, des politiques définissant les localisations géographiques et d'autres politiques définissant la bande passante requise.

Une autre solution est de faire recours à un *trader* flexible. Elle consistera à définir un ensemble de meta données compatibles avec les politiques d'adaptation du service ainsi qu'avec les caractéristiques du contexte d'exécution du service. Le modèle des meta données sera ensuite utilisé pour l'indexation et la recherche de nouvelles politiques en utilisant des ontologies de domaine (proposé dans le cadre du projet CARISM II).

Gestion des ressources de la machine Les services de compression et fragmentation sollicitent intensivement les ressources internes de la machine. Comme nous l'avons effectué pour la vitesse de transmission et l'économie d'énergie, des modèles devront établir le lien entre ces services et la consommation de ressources comme le processeur et la mémoire. Ainsi, l'utilisation de ces services devra se faire dans le cas où la machine possèderait les ressources suffisantes. Nous pouvons penser également à des modèles économiques qui associeront l'usage de ces ressources à des prix qui sont fixés suivant leur disponibilité.

Partage des ressources entre plusieurs services L'exécution simultanée de plusieurs services introduit le problème du partage des ressources de la machine. Le modèle d'adaptation devra comporter des mécanismes basés sur le mode d'exécution et la priorité de chaque service afin de partager et d'allouer les ressources qui leur sont nécessaires.

Adapter MobileJMS aux environnements diffus Enfin, la perspective la plus ambitieuse sera sans doute d'adapter MobileJMS aux environnements diffus. Ce travail posera des problématiques non triviales car JMS est destiné à la base à des environnements supposant l'existence d'une entité centralisée. Toutefois, des schémas de type pair à pair permettront à des entités MobileJMS d'interagir directement dans un réseau ad hoc. Chaque entité sera client et fournisseur en même temps. Elle pourra ainsi consommer les messages localement ou les propager vers d'autres entités. Parmi les problèmes à résoudre, citons la propagation des messages d'un hôte vers un autre ou le nommage en l'absence de toute entité centralisée.

9.4 Conclusion

MobileJMS est le résultat d'une réflexion sur les propriétés que doivent posséder les intergiciels mobiles. Il propose une gestion ouverte et extensible des problématiques des environnements nomades, à travers un modèle d'adaptation, une architecture (MOM), des composants et une algorithmique avec vérification. La communication entre les entités réparties est adaptable et configurable. L'architecture et les composants de MobileJMS ont été conçus afin d'aborder plus concrètement le problème de l'adaptation des systèmes mobiles.

L'une des idées principales de cette thèse est de répartir la logique d'adaptation entre les utilisateurs, les applications et l'intergiciel. Les préférences des utilisateurs et les besoins des applications sont prises en considération par l'intergiciel pour mettre en oeuvre des décisions d'adaptation et pour faire le compromis entre la disponibilité des ressources et la qualité du service fournie.

Nous avons préféré adopter une démarche de continuité par rapport aux technologies actuelles que celle de rupture totale. En effet, l'un des facteurs de succès des services mobiles est de fournir aux développeurs le même type d'environnements que celui qu'ils maîtrisent pour les applications classiques. Nous pensons que notre travail appellera des évolutions futures sur la base des concepts qui ont été énoncés.

Liste des publications

- M. Kaddour et L. Pautet. Towards an Adaptable Message Oriented Middleware for Mobile Environments. *IEEE ASWN'03*, Bern, Suisse, Juillet 2003.
- M. Kaddour et L. Pautet. A Middleware for Supporting Disconnections and Multi-Network Access in Mobile Environments. *IEEE 2nd Conference on Pervasive Computing (Percom).* Orlando, Florida, USA, Mars 2004.
- M. Kaddour et Laurent Pautet. Une Approche Coopérative de l'Adaptabilité des Applications Mobiles basées sur MobileJMS. *Les premières journées francophones : Mobilité et Ubiquité.* Nice, France, Juin 2004.

Bibliographie

[1] J. Abrial. Z : an introduction to formal methods. Cambridge University Press, 1995.

[2] O. Angin, A. Campbell, M. Kounavis et R. Liao. The Mobiware Toolkit : Programmable Support for Adaptive Mobile Netwoking. In *Personal Communications Magazine, Special Issue on Adapting to Network and Client Variability*. IEEE Computer Society Press, Août 1998.

[3] K. Arnold, B. O'Sullivan, R. W. Scheifler, J. Waldo et A. Wollrath. The Jini[tm] Specification. Addison-Wesley, 1999.

[4] G. Banavar, T. Chandra, R. Strom et D. Sturman . A Case for Message Oriented Middleware. In *LNCS* 1693. Springer Verlag, pages 1-18, 1999

[5] K. Barr et K. Asanovic. Energy Aware Lossless Data Compression. *The First International Conference on Mobile Systems, Applications and Services*, San Francisco, CA, Mai 2003.

[6] BEA. BEA Tuxedo. `www.bea.com/framework.jsp?CNT=index.htm\&FP=/content/products/tux`.

[7] A. D. Birrell et B. J. Nelson. Implementing Remote Procedure Calls. *ACM Transactions on Computer Systems* 2(1), 39-59, Février 1984.

[8] Blackdown.org. Java Linux. `www.blackdown.org`.

[9] G. Blair, G. Coulson, P. Robin et M. Papathomas. An Architecture for Next Generation Middleware. In *Proceedings of Middleware'98*, pages 191-206, Springer Verlag, Sept. 1998.

[10] Bluetooth.com. Bluetooth. `www.bluetooth.com`.

[11] G. A. Bolcer, M. Gorlick, A. S. Hitomi, P. Kammer, B. Morrow, P. Oreizy et R. N. Taylor. Peer-to-Peer Architectures and the Magi OpenSource Infrastructure. `www.peertopeerwgorg/downloads/collateral/proposals/ETI_peer-topeer.pdf`, Octobre 2001.

[12] E. Bruneton, T. Coupaye et J.B. Stefani. The Fractal Component Model (version 2.0-3). `fractal.objectweb.org/specification/index.html`. Février, 2004

[13] R. Caceres and L. Iftode. Improving the Performance of Reliable Transport Protocols in Mobile Computing Environments. *IEEE JSAC*, 13(5), Juin 1994.

[14] L. Capra, W. Emmerich et C. Mascolo. Reflective Middleware Solutions for Context-Aware Applications. In *Proceedings of REFLECTION 2001, The Third International Conference on Metalevel Architectures and Separation of Crosscutting Concerns*, Kyoto, Japan. Sept. 2001.

[15] L. Capra, W. Emmerich et C. Mascolo. CARISMA : Context-Aware Reflective mIddleware System for Mobile Applications. *IEEE Transactions on Software Engineering*, 29(10) :929-945, 2003.

[16] H. Chang, C. Tait, N. Cohen, M. Shapiro, S. Mastrianni, R. Floyd, B. Housel et D. Lindquist. Web browsing in a wireless environment : Disconnected and asynchronous operation in ARTour Web Express. In *Proceedings of the 3rd Annual ACM/IEEE International Conference on MobileComputing and Networking MOBICOM '97*, Budapest, Hungary, 26-30 Sept., 1997.

[17] M. Clarke, G. Blair, G. Coulson et N. Parlavantzas. An Efficient Component Model for the Construction of Adaptive Middleware. In *Proceedings of Middleware 2001*, Heidelberg, Germany, Nov. 2001.

[18] G. Coulouris, J.Dollimore et T. Kindberg. Distributed systems : Concepts and Design. Harlow, UK : Addison-Wesley, 2001.

[19] L. Deutsch. Gzip File Format Specification, 1996.

[20] L. Deutsch. DEFLATE Compressed Data Format Specification. Disponible dans `ftp://ftp.uu.net/pub/archiving/zip/doc`. 1996.

[21] A. Diller. The B-book. John Willey & SONS, 1994.

[22] J. Dorsey et D. P. Siewiorek. The Design of Wearable Systems : A Shift in Development Effort In *Proceedings of the International Conference on Dependable Systems and Networks (DSN-2003)*, San Francisco, CA, Juin 2003.

[23] B. Dumant, F. Horn, F. Dang Tran et J. B. Stefani. Jonathan : an Open Distributed Processing Environment in Java. In *Proceedings of the IFIP International Conference on Distributed Systems Platforms and Open Distributed Processing.* The Lake District, U.K., Sept. 1998.

[24] J. Flinn et M. Satyanarayanan. Energy-aware Adaptation for Mobile Applications. *Symposium on Operating Systems Principles*, 48-63, 1999.

[25] J. Flinn et M. Satyanarayanan. PowerScope : a Tool for Profiling the Energy Usage of Mobile Applications. *Second IEEE Workshop on Mobile Computing Systems and Applications*, New Orleans, LA, Pages 2-10. Fev. 1999

[26] G. H. Forman et J. Zahorjan. The Challenges of Mobile Computing. *IEEE Computer*, 27(6), Avril 1994.

[27] A. Fox, S. D. Gribble, E. A. Brewer et E. Amir. Adapting to network and client variability via on-demand dynamic distillation. In *Proceedings of the seventh international conference on Architectural support for programming languages and operating systems*, pages 160-170, ACM Press, 1996.

[28] D. Fritsch, D. Klinec et S. Volz. NEXUS Positioning and Data Management Concepts for Location Aware Applications. In *Proceedings of the 2nd International Symposium on Telegeoprocessing*, pages 171-184, Sophia-Antipolis, France, 2000.

[29] E. Gamma, R. Helm, R. Johnson et J. Vlissides. Design Patterns : Elements of Reusable Object-Oriented Software. Addison-Wesley, 1995.

[30] M. S. GAST. 802.11 Wireless Networks : the Definitive Guide. O'Reilly, 2002

[31] D. Gelernter. Generative Communication in Linda. *ACM Transactions on Programming Languages and Systems*, 7(1) :80-112, 1985.

[32] P. Grace, G. S. Blair et S. Samuel. ReMMoC : A Reflective Middleware to Support Mobile Client Interoperability. In *Proceedings of International Symposium on Distributed Objects and Applications(DOA)*, Catania, Sicily, Italy, Novembre 2003.

[33] M. Haahr, R. Cunningham et V. Cahill. Supporting CORBA Applications in a Mobile Environment. *MobiCom '99 : 5th International Conference on Mobile Computing and Networking*, Seattle, Août 1999.

[34] N. Halbwachs. A Tutorial of Lustre, 1993.

[35] Handhelds.org. Familiar Linux Project. `www.handhelds.org/geeklog/index.php`.

[36] Hapner M. et al. Java Message Service specification. Sun MicroSystems, Avril 2002

[37] G. J. Holzman. Design and validation of Computer Protocols. Prentice-Hall, 1991.

[38] G. Holzmann. SPIN Model Checker : The Primer and Reference Manual. Addison Wesley Professional, 2004.

[39] P. Honeyman et L. Huston. Communication and Consistency in Mobile File Systems. *IEEE Personal Communications*, Décembre 1995.

[40] Y. Hongying. Overview of HiperLan/2, `www.comlab.hut.fi/opetus/333/2004slides/topic-5.pdf`.

[41] B. C. Housel et D. B. Lindquist. WebExpress : A System For Optimizing Web Browsing in a Wireless Environment. In *Proceedings of the 2nd Annual International Conferenceon Mobile Computing and Networking*, ACM Press, New York, NY, 108-116. 1996

[42] N. C. Hutchinson et L. L. Peterson. The x-Kernel : An architecture for implementing Network Protocols. *IEEE Transactions on Software Engineering*, 17(1) :64-76, Jan. 1991.

[43] IBM. CICS Universal Client V6.0. `www-306.ibm.com/software/htp/cics/cuc`.

[44] IBM. WebSphere MQFamily. `www.software.ibm.com/ts/mqseries`.

[45] IBM. WebSphere EveryPlace Access. `www-306.ibm.com/software/pervasive/ws_everyplace_access`.

[46] T. Imielinski, S. Vishwanath et B. Badrinath. Energy efficient indexing on air. In *Proceedings of the 1994 ACM SIGMOD International Conference on Management of Data SIGMOD '94*, Minneapolis, MN, Mai, 1994.

[47] ISO/IEC TR 14252. Information technology - Guide to the POSIX Open System Environment (OSE). 1996.

[48] ITU-T. ODP Reference Model : Overview. ISO/IEC Recommendation X.901. International Standard 10746-1, 1995

[49] J. Jing, A. S. Helal et A. Elmagarmid. Client-Server Computing in Mobile Environments, *ACM Computer Survey* 31(2), pages 117-157, 1999.

[50] A. D. Joseph, J. A. Tauber et M. F. Kaashoek. Mobile computing with the Rover toolkit. *IEEE Trans. Comput.* 46(3) :337-352, 1997

[51] J. J. Kistler et M. Satyanarayanan. Disconnected Operation in the Coda File System, *Thirteenth ACM Symposium on Operating Systems Principles*, 25(5) :213-225, ACM Press, 1991.

[52] F. Kon, M. Roman, P. Liu, J. Mao, T. Yamane, L. Maes et R. Cambpell. Monitoring, Security, and Dynamic Configuration with the dynamicTAO Reflective ORB. In *International Conference on Distributed Systems Platforms and Open Distributed Processing (Middleware'2000)*, pages 121-143, New York, ACM/IFIP, Avril 2000.

[53] F.Kordon et E. Paviot-Adet. Using CPN-AMI to validate a Safe Channel Protocol. In *the proceedings of the International Conference on Theory and Applications of Petri Nets* - Tool presentation part, Williamsburg, USA, Juin, 1999.

[54] G. Kortuem, J. Schneider, D. Preuitt, G. Thaddeus, T. Cowan, S. Fickas et Z. Segall. When Peer-to-Peer comes Face-to-Face : Collaborative Peer-to-Peer Computing in Mobile Ad-hoc Networks. In *Proceedings 2001 International Conference on Peer-to-Peer Computing (P2P2001)*, Suède, Aout 2001.

[55] J Kramer et J. Magge. The Evolving Philosophers Problem : Dynamic Change Management. *IEEE Trans. Software Engineering*, 16(11) :1293-1306, Novembre 1990.

[56] G. H Kuenning et G. J. Popek. Automated hoarding for mobile computers. *ACM SIGOPS Operating Systems Review* 31(5) :264-275, 1997

[57] J. Lansford. HomeRF : Bringing wireless connectivity. `Home.grouper.ieee.org/groups/802/11/Tutorial/90548S-WPAN-HomeRF-Tutorial.pdf`, 1999.

[58] E. de Lara, D. Wallach et W. Zwaenepoel. Puppeteer : Component-based Adaptation for Mobile Computing. *USENIX Symposium on Internet Technologies and Systems*, Mars 2000.

[59] T. Ledoux. OpenCorba : A Reflective Open Broker. In *Reflection'99*, volume 1616 of *LNCS*, pages 197-214, Saint-Malo, France, 1999

[60] J. Lorch et A. J. Smith. Software Strategies for Portable Computer Energy Management. *IEEE Personal Communications Magazine* 5(3) :60-73, Juin 1998.

[61] C. Mascolo, L. Capra et W. Emmerich. Middleware for Mobile Computing. In *Advanced Lectures on Networking - Networking 2002 Tutorials*, Pisa, Italy. volume 2497 of LNCS, pages 20-58, Springer Verlag. Mai 2002.

[62] C. Mascolo, L. Capra, S. Zachariadis et W. Emmerich. XMIDDLE : A Data-Sharing Middleware for Mobile Computing. In *Personal and Wireless Communications Journal* 21(1), Kluwer. Avril 2002.

[63] R. Meier, V. Cahill. STEAM : Event-Based Middleware for Wireless Ad Hoc Networks. *22nd International Conference on Distributed Computing Systems Workshops (ICDCSW '02)*, Vienna, Austria, Juillet 2002.

[64] Microsoft. COM : Component Object Model Technologies. `www.microsoft-.com/com/default.mspx`.

[65] Microsoft. Microsoft Message Queuing (MSMQ). `www.microsoft.com/windows2000/technologies/communications/msmq/default.asp`.

[66] Microsoft. The Distributed Component Object Model (DCOM). `www.microsoft.com/com/tech/DCOM.asp`, 1997.

[67] S. Mohapatra and N. Venkatasubramanian. Optimizing Power using a Reconfigurable Middleware. *Middleware. Tech. rep.*, UC, Irvine, 2003.

[68] Mque Systems. Open Source Message Queue (OMSQ). `www.omsq.org`.

[69] L. Mummert et M. Satyanarayanan. Large granularity cache coherence in the coda file system. In *Proceedings of the USENIX Summer Conference* (Boston, Mass.), USENIX Assoc., Berkeley, CA., 1994.

[70] A. L. Murphy, G. P. Picco, et G.-C. Roman. Lime : A Middleware for Physical and Logical Mobility. In *Proceedings of the 21st International Conference on Distributed Computing Systems (ICDCS-21)*, Mai 2001.

[71] National Institute of Standards and Technology. NIST Net. `snad.ncsl.nist.gov/itg/nistnet`.

[72] ObjectWeb Consortium. JORAM. `joram.objectweb.org`.

[73] ObjectWeb Consortium. JONATHAN. `jonathan.objectweb.org`.

[74] OMG. Unreliable Multicast Inter-ORB Protocol Request For Proposal. OMG Document orbos/99-11-14, November 1999.

[75] OMG. Minimum CORBA, v1.0. `www.omg.org/cgi-bin/doc?formal/02-08-01`, 2002.

[76] OMG. CORBA Component Model, v3.0. `www.omg.org/cgi-bin/doc?formal/02-06-65`, 2002.

[77] OMG. CORBA Notification Service, v1.0.1. `www.omg.org/cgi-bin/doc?formal/2002-08-04`, 2002.

[78] OMG. Real-Time CORBA, v2.0. `www.omg.org/cgi-bin/doc?formal/03-11-01`, 2003.

[79] OMG. Common Object Request Broker Architecture, v3.0.2. `www.omg.org/cgi-bin/doc?formal/04-03-01`, 2004.

[80] OMG. Wireless Access and Terminal Mobility in CORBA, v1.1. `www.omg.org/cgi-bin/doc?formal/2004-04-02`, 2004.

[81] Openjms.org. OpenJMS project. `www.openjms.org`.

[82] T. Quinot. Conception et Réalisation d'un Intergiciel Schizophrène pour la mise en oeuvre de Systèmes Répartis Interopérables. *Thèse de doctorat* (24 mars 2003), Groupe ASTRE/département Infres/École nationale supérieure des télécommunications — LIP6/Université Pierre-et-Marie-Curie.

[83] D. Ratner, P. L. Reiher, G. J. Popek et G. H. Kuenning. Replication Requirements in Mobile Environments. *Mobile Networks and Applications*, 6(6) :525-533, ACM Press, 2001.

[84] D. Ratner, P. Reiher et G. J. POPEK. Roam : A Scalable Replication System for Mobility. *Mobile Networks and Applications*, 9(5) :537-544, ACM Press, 2004.

[85] P. Romano, M. Romero, B. Ciciani et F. Quaglia. Validation of the Sessionless Mode of the HTTPR Protocol. *The 23rd IFIP International Conference on Formal Techniques for Networked and Distributed Systems*, FORTE 2003, Berlin (Germany), Oct. 2003.

[86] Saha, D., and Mukherjee, A.Pervasive Computing : A Paradigm for the 21st Century, *Computer*, 36(2) :25-31, IEEE Computer Society Press, 2003.

[87] Salutation Consortium. Salutation. `www.salutation.org`, 1999.

[88] M. Satyanarayanan. Pervasive Computing : Vision and Challenges. *IEEE Personal Communications*, pages 10-17, Aout 2001.

[89] A. Schill, W. Bellmann et S. Kummel. System Support for Mobile Distributed Applications. *2nd International Workshop on Services in Distributed and Networked Environments*, Whistler, British Columbia, Juin 1995.

[90] H. Schulzrinne, S. Casner, R. Frederick et V. Jacobson, RTP : A Transport Protocol for Real-time Applications. *RFC 1889*, Internet Engineering Task Force, Jan. 1996.

[91] A. Smith. Reflection and Semantics in a Procedural Programming Langage. *Phd thesis*, MIT, Jan. 1982.

[92] Softwired. iBus//mobile. `www.softwiredinc.com/products/mobile/mobile.html`.

[93] SQL92 Syntax. `owen.sj.ca.us/rkowen/howto/sql92F.html`.

[94] Sun Microsystems. Java RMI specification, 1998.

[95] Sun Microsystems. Java Architecture for XML Binding (JAXB). `java.sun.com/xml/jaxb`, 2003.

[96] Sun Microsystems. Entreprise JavaBeans Technology. `java.sun.com/products/ejb`, 2004.

[97] Sun Microsystems. Java 2 Micro Edition. `java.sun.com/j2me`.

[98] C. Szyperski. Component Technology : What, Where, and How ? In *Proceedings of the 25th international conference on Software engineering*, Portland, Oregon, Mai 2003.

[99] D. L. Tennenhouse et D. J. Wetherall. Towards an Active Network Architecture. *Computer Communication Review*. 26(2), 1996.

[100] A. Terry, M. Theimer, K. Petersen, A. Demers, M. Spreitzer, et C. Hauser. Managing Update Conflicts in Bayou, a Weakly Connected Replicated Storage System. In *Proceedings of the 15th ACM Symposium on Operating Systems Principles (SOSP-15)*, pages 172-183, Cooper Mountain, Colorado, Aout. 1995.

[101] Tibco. Tibco Rendezvous. `www.tibco.com/software/enterprise_backbone/rendezvous.jsp`.

[102] Ubi-core. Universally Interoperable Core. `www.ubi-core.com`, 2001.

[103] UPnP Forum. Universal Plug and Play. `www.upnp.org`, 1998.

[104] J. Veizades, E. Guttman, C. Perkins et S. Kaplan. Service Location Protocol, RFC 2165. `www.ietf.org/rfc/rfc2165.txt`, Juin 1997.

[105] T. Vergnaud, J. Hugues , L. Pautet et F. Kordon. PolyORB : A Schizophrenic Middleware to build Versatile Reliable Distributed Applications. In *Proceedings of the 9th International Conference on Reliable Software Techologies Ada-Europe 2004 (RST'04)*, volume LNCS 3063, pages 106 - 119, Palma de Mallorca, Spain, Juin 2004.

[106] Wap Forum. Wireless Markup Language Specification, Version 1.2. 1999.

[107] W3C. Document Object Model (DOM) Level 1 Specification. `www.w3.org/TR/1998/REC-DOM-Level-1-19981001`, Octobre 1998.

[108] W3C. XML Path Language (XPath) Version 1.0. `www.w3.org/TR/xpath`, 1999

[109] W3C. Web Services Description Language (WSDL) v1.1. `www.w3.org/TR/2001/NOTE-wsdl-20010315`, Mars 2001.

[110] W3C. XML Linking Language (XLink) Version 1.0. `www.w3.org/TR/2001/RECxlink-20010627`, 2001.

[111] W3C. XHTML 1.0 The Extensible HyperText Markup Language. `www.w3.org/TR/xhtml1`, 2002.

[112] W3C. CC/PP Information Page. `www.w3.org/Mobile/CCPP`.

[113] W3C. Resource Description Framework (RDF). `www.w3.org/RDF`.

[114] J. Waldo. JavaSpaces, Specification 1.0. Technical Report, Sun Microsystems. Mars 1998

[115] P. Wyckoff, S. W. McLaughry, T. J. Lehman et D. A. Ford. T Spaces. *IBM Systems Journal*, 37(3) :454-474, 1998.

[116] R. Xu, Z. Li, C. Wang et P. Ni. Impact of Data Compression on Energy Consumption of Wireless-Networked Handheld Devices. *ICDCS '03 : Proceedings of the 23rd International Conference on Distributed Computing Systems*, page 302, IEEE Computer Society, 2003.

[117] E. Yoneki et J. Bacon. Pronto : Messaging Middleware over Wireless Networks. In *Proceedings of ACM/IFIP/USENIX International Middleware Conference (Work in Progress).* Rio de Janeiro, Brazil, Juin 2003.

[118] ZigBee Alliance. `www.zigbee.org`.

[119] J. Ziv et A. Lempel. A Universal Algorithm for Sequential Data Compression. *IEEE Transactions on Information Theory*, 23(3) :337-343, 1977