VERS UNE INTÉGRATION TRANSPARENTE DE L'INFORMATIQUE DANS NOTRE ENVIRONNEMENT

Application à un système de rappels informatique diffus

Par

Simon Guertin

mémoire présenté au Département d'informatique en vue
de l'obtention du grade de maître ès sciences (M.Sc.)

FACULTÉ DES SCIENCES
UNIVERSITÉ DE SHERBROOKE

Sherbrooke, Québec, Canada, février 2005

Le 7 mars 2006

le jury a accepté le mémoire de M. Simon Guertin dans sa version finale.

Membres du jury

M. Sylvain Giroux
Directeur
Département d'informatique

M. Philippe Mabilleau
Membre
Département de génie électrique et informatique

M. Gabriel Girard
Président-rapporteur
Département d'informatique

SOMMAIRE

Un des projets de recherche du laboratoire DOMUS vise à réaliser une maison intelligente qui fournit une aide cognitive à des personnes en légère perte d'autonomie. La recherche présentée dans ce mémoire vise principalement à réaliser un système informatique diffus qui intègre des réseaux, appareils électriques et systèmes informatiques, tous hétérogènes, dans le but de coopérer pour réaliser une tâche qui sera bénéfique à l'occupant des lieux. Le système se base sur les principes de l'informatique diffuse pour intégrer l'informatique à l'environnement physique. Nos travaux développent une étude de cas visant à implémenter un système de messagerie de notification et de rappel dans un environnement informatique diffus. Nos travaux ont atteint les buts visés en réalisant trois scénarios qui illustrent différents aspects de l'informatique diffuse. Le premier scénario vise la détection et l'adaptation du système face à la présence de l'usager lorsque celui-ci entre dans une pièce. Le deuxième scénario réalise la migration d'une session dans une application d'un appareil vers un autre appareil qui est mieux adapté au besoin de l'usager. La session dans l'application du premier appareil migre sur un autre appareil et est récréée le plus fidèlement possible pour ne pas que l'usager ait à refaire les mêmes étapes deux fois de suite. Le troisième scénario représente une tentative par le système de localiser l'usager pour lui transmettre un message. Ce scénario implique une grande coopération entre divers services qui sont disponibles dans l'architecture Domus.

REMERCIEMENTS

Je remercie mon directeur de recherche, le professeur Sylvain Giroux pour sa direction, son aide et son soutien. Je remercie aussi la professeure Hélène Pigot qui m'a fait voir d'autres facettes de l'informatique lorsque j'ai collaboré avec elle en tant que chargé d'exercices et lors de la conférence ICOST 2004. Je remercie le département d'informatique pour ce qu'il m'a apporté académiquement, les rencontres et les liens d'amitiés qui se sont tissés avec d'autres étudiants du département et de la communauté universitaire. Je remercie mon père Jules, ma mère Carmen-Andrée, ma soeur Marie-Pierre et mes amis pour leurs encouragements et leurs appuis tout au long de mes études.

TABLE DES MATIÈRES

LISTE DES ABRÉVIATIONS

- PDA, PDAs : Personal Digital Assistant. Ordinateur personnel de poche.
- VNC : Virtual Network Computing. Application client-serveur qui permet de contrôler un autre ordinateur à distance. L'usager qui prend le contrôle de l'ordinateur distant a accès à une interface graphique identique à celui qui est présent sur l'ordinateur hôte.
- MVC : Model-View-Controller. Patron de conception qui consiste à séparer les données et l'interface graphique dans une application en utilisant un contrôleur qui fait le lien et les mises à jour entre les deux entités.

LISTE DES FIGURES

INTRODUCTION

Avec le coût des ordinateurs qui diminue toujours, la puissance des processeurs qui augmente sans cesse, la puissance de traitement des appareils portables (appareils cellulaires, PDAs[1]) qui augmente toujours et l'amélioration des dispositifs d'affichage de ces derniers, les réseaux sans-fil qui sont de plus en plus répandus et l'apparition de nouveaux langages de programmation, une nouvelle ère de l'informatique s'ouvre. On peut enfin réaliser des systèmes informatiques diffus. Depuis de nombreuses années, nous pouvons connecter entre elles plusieurs composantes de chaînes stéréo et audio vidéo. Seulement en connectant les bons fils en entrée et en sortie, sans aucun besoin de configuration et le système fonctionne à merveille, sans devoir acheter les composantes chez la même compagnie ou durant la même année. Ce n'est pas encore le cas dans le monde de l'informatique, mais c'est envisageable avec les nouveaux outils disponibles. Dans l'avenir, les ordinateurs devront être intégrés parfaitement à notre environnement, voire invisibles. Nos tâches quotidiennes feront appel à des ordinateurs sans s'en rendre compte, comme nous nous servons d'un ascenseur, d'une chaise ou d'une feuille de papier pour écrire. La maison informatisée sera réalisée d'une manière répartie, sans serveur centralisé, où les composantes apparaissent et disparaissent du système et utilisent d'autres appareils pour réaliser leurs tâches sans l'intervention d'un utilisateur. Le système est plus robuste et nécessite une configuration minimale, idéalement non existante. Il peut y avoir des applications centrales qui peuvent accéder toutes les composantes du système pour de la configuration ou du diagnostic, mais le système doit fonctionner sans ces applications, de manière autonome.

Dans le cadre du projet DOMUS de l'Université de Sherbrooke, on doit réaliser un système informatique diffus (*ubiquitous*). Le projet DOMUS consiste en un laboratoire informatique qui réalise un appartement informatisé qui comporte de nouveaux concepts en matière d'informatique par rapport à la personne. Ce projet regroupe plusieurs disciplines :

[1] **P**ersonal **D**igital **A**ssistant

informatique, éthique, psychologie et santé. Les domaines d'applications de ce laboratoire sont la santé, l'éducation, la protection civile et le tourisme. Ce système se compose de plusieurs appareils informatiques, appareils sans-fil et d'appareils électriques. Le lieu où est installé le système sera adapté automatiquement à l'usager afin de lui fournir un environnement plus confortable et adapté à ses activités. Les appareils collaboreront entre eux pour informer l'occupant de l'arrivée de n'importe quel message s'il y a lieu de le faire. Le système pourra aussi détecter des oublis de l'usager comme le four qui reste allumé durant des heures ou des anomalies dans le comportement de l'usager. Dans le projet, nous utilisons un agenda électronique qui essaie de prévenir l'usager qu'un rendez-vous approche. La maison tentera par plusieurs moyens, qui peuvent changer selon les circonstances, d'avertir l'usager. Plusieurs aspects doivent être abordés pour réaliser le projet. Une première caractéristique est la position de l'usager et la position des appareils les uns par rapport aux autres. L'apparition spontanée et la disparition des services et des appareils ne doivent pas affecter le fonctionnement global du système. Une application peut aussi migrer de manière paramétrique d'un appareil vers un autre. La migration ne s'effectue pas au niveau des processus mais bien au niveau d'où est rendu l'utilisateur dans l'application. La migration de l'application est réalisée par la sauvegarde de l'état des fenêtres (contenu, position) utilisées par l'utilisateur. Avec ces données, l'application peut être reconstruite sur un autre ordinateur. Selon la situation, l'usager peut vouloir utiliser un appareil avec un meilleur affichage pour continuer à travailler. Il peut aussi vouloir passer d'un appareil fixe vers un appareil sans-fil s'il veut se promener tout en travaillant dans l'application. Lorsqu'il changera d'appareil, il aura à effectuer un minimum d'étapes pour continuer là où il avait quitté sa session sur l'ancien appareil. Un appareil peut aussi avoir besoin d'un autre appareil plus puissant pour pouvoir réaliser sa tâche. Le logiciel devra découvrir spontanément un autre service qui correspond aux besoins recherchés. Une des caractéristiques principales abordées dans ce projet est l'hétérogénéité des appareils informatiques et électriques qui devront travailler tous ensemble. Ces appareils n'ont pas étés conçus dans le but d'effectuer ce travail et il faudra trouver des solutions pour rendre le système le plus homogène possible. Un aspect difficile à gérer sera la détection et

l'identification des personnes dans une pièce avec les moyens qui seront à notre disposition afin que le système soit mis au courant de leur présence.

Le contenu de ce mémoire présente la réalisation d'un système informatique diffus d'agenda. L'agenda diffus est un agenda électronique qui a pour fonction première d'emmagasiner des rendez-vous, par contre, sa fonctionnalité très particulière est son intégration et son utilisation dans le système complet. Cette fonctionnalité met en oeuvre beaucoup d'autres systèmes pour effectuer des tâches collaboratives afin d'informer l'usager qu'un rendez-vous approche. Ce système s'appuie sur trois scénarios d'utilisation qui constituent une base pour la réalisation d'un système complet d'assistance cognitive utilisant divers aspects de l'informatique diffuse. Dans sa réalisation sont intégrés des appareils informatiques, électriques, électroniques, ainsi que des réseaux hétérogènes. Dans ce mémoire, nous présentons d'abord la situation et le contexte de la recherche (Chapitre 1), le projet et son laboratoire d'expérimentation (Chapitre 2) et les technologies utilisées (Chapitre 3). Par la suite nous attaquons la réalisation du projet en montrant l'architecture logicielle (Chapitre 4), la description des scénarios d'utilisation (Chapitre 5), pour terminer avec une comparaison du système avec d'autres réalisations (Chapitre 6) et la conclusion.

CHAPITRE 1

ÉTAT DE L'ART

La recherche en informatique diffuse en est encore à ses débuts. Nous présentons ici le contexte de recherche et les besoins des systèmes informatiques diffus. Le contexte de recherche est présenté via la critique de trois articles qui traitent de l'informatique diffuse en général (1.1 The Computer for the 21st Century [19]), des besoins pour réaliser des systèmes informatique diffus (1.2 System Software for Ubiquitous Computing [10]) et d'une solution qui a été réalisée (1.3 Using Pervasive Computing to Deliver Elder Care [15]).

1.1 L'ordinateur au 21ième siècle

Dans l'article « The Computer for the 21st Century » [19], Mark Weiser, en introduisant les concepts de l'informatique diffuse ('ubiquitous computing' en anglais), explique comment la technologie peut réussir à disparaître. Cette technologie réussit à disparaître en s'intégrant profondément à l'environnement. L'usager n'a pas à utiliser directement cette technologie, mais pourtant elle réagit constamment aux actions de l'usager. Elle peut modifier l'environnement et s'adapter aux conditions changeantes sans l'intervention de l'usager. L'usager n'a pas besoin de l'utiliser directement, mais la technologie est présente et prête à être utilisée. Elle devient alors partie intégrale de notre environnement,

> « The most profound technologies are those that disappear. They weave themselves into the fabric of everyday life until they are indistinguishable from it. ». [19].

Malgré toutes les ventes d'ordinateurs, la technologie des ordinateurs ne réussit toujours pas à se confondre à son environnement. Il faut penser à une nouvelle manière de voir les ordinateurs. Il faut réussir à les intégrer à notre environnement pour qu'ils disparaissent en arrière plan. Cette disparition de la technologie n'est pas une conséquence de la

technologie, mais une conséquence de la psychologie humaine. Quand on comprend une chose assez bien, on l'utilise sans y penser pour atteindre de nouveaux buts. C'est ce qui arrive lorsqu'on lit un panneau de signalisation. Lors de la lecture, on comprend et intègre l'information que le panneau contient sans pour autant exécuter une tâche de lecture. *« Ubiquitous computing »* ne signifie donc pas que l'on peut apporter notre ordinateur partout même s'il est très puissant car notre attention sera toujours portée sur la machine elle-même. Comment les technologies se dissimulent-elles dans notre environnement ? Un bon exemple serait celui de l'automobile qui est constituée de plusieurs petits moteurs qui actionnent plusieurs systèmes de la voiture. Il y a deux facteurs d'importance pour la dissimulation des ordinateurs : l'endroit et la taille. Les ordinateurs doivent savoir où ils se trouvent physiquement pour pouvoir mieux interagir avec les autres ordinateurs. Les ordinateurs « *ubiquitous* » viennent en plusieurs formats pour répondre à des tâches spécifiques et l'auteur en a construit trois avec ses collègues : le premier, grand comme un Post-It (un tab), le deuxième, grand comme une feuille de papier (un pad) et le troisième, de la taille d'un tableau. Le pad est composé d'un microprocesseur, de batteries, d'un bouton de contrôle et d'émetteur infrarouge. Une salle peut contenir un très grand nombre de ces ordinateurs, ce qui peut être intimidant sauf si cette technologie devient transparente pour effectuer les tâches quotidiennes. L'équipe de Weiser a conçu et réalisé un tab qui incorpore un émetteur, identifie le porteur et le situe dans un édifice. On peut aussi minimiser une application d'un ordinateur sur le tab pour la maximiser plus tard sur un autre ordinateur. Les programmes et les fichiers minimisés sur les tabs peuvent donc être restaurés sur n'importe quel terminal. L'article ne fait pas mention de la manière dont les applications sont emmagasinées sur les pads. Ceci libère ainsi l'écran de l'ordinateur d'icônes d'application minimisées. Grâce à ce tab, les portes de l'édifice s'ouvrent, les ordinateurs réagissent aux usagers, la secrétaire peut savoir où sont les personnes. Les pads sont des ordinateurs que l'on ne doit pas emporter avec nous. On doit s'en servir mais les laisser dans la pièce. Ils ne doivent pas avoir d'importance ou d'identité. On peut mettre plusieurs pads côte à côte pour couvrir un bureau comme on le ferait avec des feuilles de papiers. Les tableaux sont utilisés avec une craie sans-fil et la barre de commandes est au bas de l'écran pour les plus petites personnes qui n'atteignent pas le haut du tableau. On

n'a pas besoin de les activer ou de prévenir quelqu'un pour les utiliser, ils sont placés un peu partout. Ces trois types d'ordinateurs constituent le début de l'informatique diffuse, mais la vraie puissance ne vient pas de ces ordinateurs en eux-mêmes mais de leurs interactions.

La technologie nécessaire à l'informatique diffuse repose sur les trois champs suivants : des ordinateurs abordables, des logiciels pour l'informatique diffuse et un réseau pour connecter le tout. Malgré la faisabilité accrue de l'informatique diffuse vers la fin de la décennie, la tendance actuelle en systèmes distribués est de concevoir les serveurs de fichiers et les imprimantes comme connectés directement à un ordinateur indépendamment de leur location. Cette configuration ne prend pas avantage de la location physique des appareils. Réalité d'aujourd'hui : la technologie existe pour réaliser les tâches mais certains composants sont encore trop coûteux, même si le coût des ordinateurs personnels est rendu très bas [12]. Par contre, les réseaux sans-fil et filaires sont rendus assez développés et abordables pour réaliser le système [12]. Du point de vue logiciel, on peut réaliser des logiciels pour l'informatique diffuse avec Java [12] et Jini [18]. Le design des systèmes d'exploitation s'attend à ce que la configuration matérielle ne change pas, mais dans un système informatique diffus, les appareils disparaissent et sont ajoutés à tout moment. La migration des applications reste toujours problématique. Le réseau qui connecte tous les appareils pose aussi un problème. Contrairement au moment où Weiser a écrit cet article (1991), aujourd'hui la distance des connexions sans-fil ne pose plus un problème et constitue un avantage dans l'interaction des usagers avec leur environnement. Avec la limitation de la distance, le système va pouvoir s'adapter et trouver des services qui sont plus près physiquement de l'usager. Les appareils sans-fil peuvent maintenant avoir un rayon d'opération de 10m (Bluetooth [2] [8]), 100m (réseau local sans-fil) ou de plusieurs kilomètres (téléphones cellulaires numériques). Donc, on peut choisir la portée qui sera la plus utile pour une application donnée. Pour ce qui est du nombre limité de connexions, la technologie Bluetooth permet jusqu'à 7 connexions simultanées et les réseaux sans-fil (Wiski 802.11b) supporte une bande passante totale de 11Mbps par point d'accès peu importe le nombre d'usagers. Pour avoir une plus grande bande passante, il y a

les technologies Wi-Fi 802.11a et Wi-Fi 802.11g qui fournissent 54Mbps ou plus de bande passante. La technologie est assez flexible pour développer un système avec un assez grand nombre d'appareils sans-fil. Ainsi, avec le mélange des différentes technologies, nous avons une grande marge de manœuvre pour développer un bon système. Dans son article, Mark Weiser présente ensuite ce que pourrait être une journée typique où l'informatique diffuse serait parfaitement intégrée à notre environnement. Finalement, deux problèmes amenés par l'informatique diffuse sont l'atteinte à la vie privée et la sécurité des transmissions. Toutefois, nous pouvons affirmer que de nos jours, les ordinateurs sont de plus en plus intégrés à l'environnement et que la réalisation de système informatique diffus devient de plus en plus à notre portée.

1.2 Le logiciel et l'informatique diffuse

L'article intitulé « System Software for Ubiquitous Computing » [10] discute des caractéristiques d'un système informatique diffus. Il aborde les différents aspects de l'informatique diffuse et élabore les caractéristiques qu'elle devra respecter. Les deux principales caractéristiques du système sont l'intégration physique (§1.2.1) et l'inter-opération spontanée (§1.2.2). L'inter-opération spontanée signifie que dans le projet, les composantes matérielles et logicielles interagiront entres-elles de manière spontanée. Il n'y aura pas de configuration à faire, et les composantes pourront travailler seules ou en équipe avec d'autres composantes si elles sont disponibles. Les autres principaux sujets abordés sont les suivants : la découverte de services et l'interaction (§1.2.3), l'adaptation (§1.2.4), l'intégration (§1.2.5), la structure du système (§1.2.6), la robustesse (§1.2.7) et la sécurité (§1.2.8). Le projet Domus intégrera tous ces points sauf la sécurité qui, à elle seule, est trop vaste. Il faut absolument aborder tous ces sujets pour développer un système diffus car ils forment un tout et en laisser un seul de côté fera en sorte que le système ne sera pas fonctionnel. Seul l'aspect de la sécurité peut être laissé de côté. Un système informatique diffus doit être utile dans la vie de tous les jours et doit aussi pouvoir opérer dans un environnement continuellement changeant. La recherche sur les systèmes informatiques

diffus a beaucoup progressé, mais il reste encore une différence significative entre les spécifications et les systèmes qui implantent ces spécifications.

1.2.1 Intégration Physique

L'intégration physique des appareils qui composent un système informatique diffus se fait dans les pièces d'une bâtisse et dans des objets d'utilisation courante. Les éléments qui composent les systèmes informatiques diffus sont présents dans des objets qui ne sont pas mobiles et d'autres qui sont mobiles. On doit considérer qu'il n'y a pas un seul système informatique diffus, mais bien plusieurs systèmes informatiques diffus qui ont chacun leurs frontières et dont certains appareils peuvent se promener d'un système vers un autre. Par exemple, un usager qui possède un ordinateur personnel de poche va se déplacer de son domicile vers son bureau. L'ordinateur personnel de poche va donc quitter le système informatique diffus du domicile pour s'intégrer au système informatique diffus du lieu de travail. Les frontières d'un système informatique diffus ne sont pas toujours délimitées par un espace physique, mais aussi par une limite de l'application du système (une frontière logique).

1.2.2 L'inter-opération spontanée

L'inter-opération spontanée des appareils signifie que, dans un environnement, les appareils doivent s'adapter dynamiquement aux changements des conditions sans reconfiguration ou à l'installation de nouveaux logiciels. Les défis au point de vue du logiciel sont grands. Avec l'intégration physique et l'inter-opération spontanée, le développeur doit faire fonctionner son application dans le système réparti dans toutes les situations et ce, sans intervention de l'usager ou presque. L'usager ne verra plus l'environnement comme une multitude d'ordinateurs connectés mais plutôt comme un environnement qui offre plusieurs services. Lors du développement d'un système, il faut prendre en considération que certaines décisions doivent être prises par des humains et

d'autres par l'ordinateur. On appelle cette frontière dans la prise de décision le Rubicon[2] sémantique. Il s'agit de la division entre le système et l'usager lors de la prise de décision de haut niveau. Lorsque la responsabilité de la décision passe du système à l'usager ou vice-versa, on dit que la frontière du Rubicon sémantique a été traversée. Lorsque plusieurs choix s'offrent au système et que celui-ci est apte à prendre une décision, la frontière n'est pas traversée. Lorsque le système ne peut plus prendre la décision à cause d'un nouvel état du système et qu'il doit déléguer la décision à l'usager, la frontière est traversée. Certaines décisions peuvent migrer du coté humain ou du coté machine, mais la définition de ces décisions doit être claire. Plusieurs chercheurs dans le domaine des systèmes informatiques diffus ont établi des scénarios sur les sujets à prendre en considération quand on crée un système informatique diffus [10].

1.2.3 La découverte et l'interaction

La découverte d'un service découle d'une application cliente qui a besoin d'accomplir une tâche qu'elle est incapable de faire par elle-même. Il doit donc rechercher un autre service apte à effectuer cette tâche. Il faut donc mettre en place un protocole pour permettre aux services de s'afficher afin que les autres services ou applications clientes puissent les découvrir et les utiliser en cas de besoin. Dans la découverte d'un service, la partie initialisation et connexion au système ne pose en général aucun problème. Par contre la localisation d'un service spécifique soulève souvent des problèmes. Les systèmes comme JINI [17] offrent toute une gamme d'outils et d'API pour supporter la publication, la découverte et l'interaction des services. Un aspect important de la découverte est la norme utilisée pour décrire un service. Un appareil qui diffuse son service et un autre qui le recherche doivent utiliser la même norme afin que la tâche de recherche soit réalisable. Il est possible d'utiliser la notion de service à un haut niveau d'abstraction. Toutefois il faut éviter toute ambiguïté dans l'énoncé de ces services. Par exemple, si un service cherche à imprimer de l'information (que ce soit afficher l'information à l'écran ou de l'imprimer sur du papier), il cherchera un service répondant à une description telle que

[2] Fleuve en Italie où Jules César s'arrêta longuement afin de réfléchir avant d'y faire traverser ses armées.

«serviceType=printing». Si une imprimante a comme description «service=print», elle ne sera pas trouvée. Par contre, le service trouvera un cadre digital[3] dont la description est «serviceType=printing», alors que le but est d'imprimer l'information et non pas seulement de l'afficher à l'écran. Le cadre digital à comme unique fonctionnalité d'afficher des images ou du texte et ne peut donc pas imprimer ces dernières sur du papier. Le service qui voulait imprimer l'information ne sait donc pas qu'il commet une erreur en utilisant le cadre digital car la description de ce dernier correspond à ce qu'il recherche. Les frontières lors de la découverte doivent aussi être spécifiée, mais là aussi les règles de découverte peuvent ne pas s'appliquer dans tous les cas. La position physique de certains services peut être importante, e.g. un service d'imprimante, tandis que pour d'autres services, le lieu n'est pas pertinent, e.g. un service de stockage de données. On doit parfois laisser l'utilisateur décider. Une fois le service découvert, l'interaction avec celui-ci pose un problème. Le service client (le service qui a besoin d'afficher une information) doit a priori savoir comment interagir avec le service voulu (le service trouvé qui fournit ce dont on a besoin comme le cadre digital cité précédemment), sinon il ne saura pas comment l'utiliser (il ne saura pas comment dire au cadre digital d'afficher une information).

Les systèmes « Event » [7], [18] et « Tuple Space » [4] offrent une approche alternative à la découverte directe et immédiate de services. Avec la première approche, on devait savoir à l'avance comment interagir avec les services tandis qu'avec celle-ci, on n'a pas besoin de connaître à l'avance la manière d'interagir avec le service. Ces deux systèmes possèdent entre autre deux caractéristiques communes : les interfaces au système ont un nombre limité et fixe d'opérations et les interactions sont définies par les données. Le système « Tuple Space » fourni un espace pour emmagasiner des tuples. Les composants du système communiquent entre eux en y déposant des tuples qui représentent soit des données, soit des processus en attente d'être évalué. Les composants peuvent prendre les tuples et en ajouter, mais un tuple ne peut pas être modifiés pendant qu'il est dans le système partagé. Dans le système « Event », les composants publient leur description dans un annuaire et ils s'abonnent à des services dont ils décrivent les propriétés. Lorsqu'un

[3] Un cadre digital est un appareil qui ressemble à un cadre où l'on remplace la glace par un écran plat.

nouveau service correspond à un service recherché, le système avertit les composants qui se sont enregistrés à ce type de service. Dans chacun des deux systèmes, les composantes interagissent entre elles en utilisant un service commun (les tuples ou bien un service «Event»). L'application cliente n'a donc pas besoin d'avoir une connaissance directe du service avec lequel elle interagit. L'interaction orientée donnée semble une bonne approche pour l'interaction spontanée, par contre elle nécessite une uniformité des données diffuses pour fonctionner dans des environnements qui comportent des frontières.

1.2.4 L'adaptation

Dans un système diffus, les appareils doivent s'adapter aux conditions changeantes de leur environnement sans intervention humaine. Les capacités de certains appareils sont plus limitées que d'autres, ce qui rend plus difficile leur adaptation à des environnements en mouvance. Le contenu d'une application doit aussi s'adapter aux différents appareils sur lesquels il peut s'afficher. Dans l'esprit du patron « Model-View-Controller » (MVC), la séparation entre l'application et l'interface usager doit être claire pour pouvoir afficher les informations sur des appareils de différentes spécifications. Le but étant de pouvoir interagir avec un service et que cette interaction réussisse à être adaptée à l'appareil utilisé.

Pour supporter et réaliser des processus adaptatifs, il y a quatre niveaux possibles d'intelligence de l'interface graphique qui est utilisée sur un appareil. Le premier niveau consiste à l'affichage des données et la saisie des interactions de l'usager avec l'interface graphique. Cette méthode ne sait pas comment les composantes graphiques interagissent avec les données. Un exemple de système qui applique cette méthode est Virtual Network Computing (VNC) [13]. Ce système client/serveur reproduit intégralement le bureau de l'usager dans une fenêtre sur un autre système, peu importe le type du système d'exploitation. Il utilise un protocole pour définir les fenêtres, les couleurs et la position de l'écriture et des rectangles.

Le second niveau utilise une description de haut niveau de l'interface usager. X Windows en est un bon exemple. Pour afficher les fenêtres, le serveur X Windows doit fonctionner sur le client. Ce niveau utilise une description de haut niveau des composantes de l'interface usager. Le serveur X Windows doit fonctionner sur le terminal de l'utilisateur pour pouvoir recréer le bureau de l'usager et permettre les interactions. Le serveur X Windows qui est exécuté sur le terminal client est surnommé serveur même si cela porte à confusion car ce dernier fonctionne avec le vrai serveur X Windows qui est exécuté sur un ordinateur central.

Le troisième niveau contient une description de l'interface qui est exportée chez le client. À partir de cette description, on peut ensuite générer l'interface usager. C'est-à-dire que l'on décrit l'interface graphique d'une manière assez générale, mais précise pour conserver les fonctionnalités. Cette description, qui se doit d'être la plus légère possible, est ensuite envoyée au client qui l'interprète pour recréer le plus fidèlement possible ce que cette dernière est supposée représenter. Le système ICrafter [11] travaille de cette manière. Les services fournissent une description de l'interface en XML et un générateur d'interfaces graphiques interprète la description et crée une interface générique ou adaptée à un appareil en particulier. Ces services peuvent être des appareils comme une lampe ou un projecteur, ou bien des applications comme un navigateur Web ou l'application Microsoft PowerPoint projetée sur un grand écran. L'usager peut interagir avec ces services en utilisant un appareil qui possède des fonctionnalités d'entrée/sortie. Ces appareils peuvent être des ordinateurs portables ou des PDAs. Chaque service exporte au générateur d'interface graphique une description de son interface. Le générateur peut alors générer pour un appareil spécifique une interface graphique pour interagir avec le service. Lorsque qu'un appareil demande au générateur une interface pour contrôler un service, l'appareil fournit aussi une description qui contient des informations à propos de ces capacités. Le générateur d'interface peut alors générer la bonne interface qui représente le service en fonction de l'appareil sur lequel il sera contrôlé. L'interface générée est une page HTML qui sera interprétée par un navigateur Web sur l'appareil utilisé pour contrôler le service.

Finalement, le quatrième niveau correspond à du code mobile téléchargé par le client et exécuté sur l'appareil hôte pour instancier l'interface. L'architecture Jini, réalisée en Java, exprime bien cette approche. L'interface est fournie par le service et le code est transféré sur l'appareil de l'usager qui a besoin d'utiliser l'interface. Ce code est ensuite exécuté sur l'appareil qui a besoin du service.

1.2.5 Intégration

Un autre aspect important dans la réalisation de système informatique diffus est l'intégration des senseurs et des appareils physiques dans l'environnement virtuel. Une couche de bas niveau est utilisée pour masquer les détails tout en continuant à refléter la nature du senseur ou de l'appareil. Cette abstraction des appareils doit être uniforme pour faciliter l'intégration des composants. Cette représentation logique permet de faire le pont entre la réalité et le monde virtuel informatique.

1.2.6 Le cadre de programmation

Les systèmes et les langages évoluent. Pourtant il ne faut pas devoir réécrire à chaque fois les systèmes spécialisés prévus pour opérer dans un environnement diffus. On doit pouvoir faire évoluer les applications diffuses déjà existantes en leur apportant un minimum de modifications pour qu'elles fonctionnent dans un nouvel environnement. Il est impossible de prévoir toutes les machines et tous les systèmes sur lesquels ils s'exécuteront. Les vieux composants devront être intégrés, même si leur technologie est désuète. C'est pourquoi il faut développer le cadre de programmation et des structures de programmes qui feront en sorte que les applications peuvent fonctionner sur des nouveaux et anciens systèmes. Il faut qu'il y ait un moyen de communication entre les systèmes incompatibles. Le projet «Stanford Interactive Workspace» [9] utilise un mécanisme de coordination au niveau applicatif par un espace de tuples. Cette approche résulte en un système à faible couplage qui est facile à adapter à de vieux systèmes.

1.2.7 Robustesse

Dans un système diffus, les sources de défaillance sont très nombreuses et variées : batteries déchargées pour les petits appareils, perte de connexion au réseau d'un appareil sans-fil parce qu'il a quitté les lieux, etc. Il faut être en mesure de récupérer le plus rapidement possible d'erreurs qui surviennent dans le système pour que le tout continue d'opérer sans l'intervention de l'usager. On peut utiliser la communication de groupe pour retrouver des ressources perdues mais en tenant compte des principes de frontières pour les systèmes diffus. Ainsi, les services peuvent être groupés en fonction de leur position physique, par exemple la cuisine. Un usager pourra ainsi effectuer les recherches de services en ne s'adressant qu'aux services appartenant au groupe « cuisine ». Les services peuvent, avec Jini, être organisés en fédération. Dans l'architecture Jini, les services ont accès à un système de baux et de renouvellement de baux pour utiliser des ressources. Les services ont aussi accès à un service de transaction pour conserver l'intégrité des données, pour ne pas que le système atteigne un état incohérent des données. Un état incohérent des données peut être atteint au niveau applicatif et aussi au niveau du système. Un état incohérent des données au niveau du système peut être atteint s'il y a des problèmes de synchronisation entre les applications qui utilisent et modifient des données partagés ou des ressources. Par exemple, un service qui s'exécute sur appareil mobile n'est plus disponible si ce dernier quitte les lieux du système diffus. L'état du système devient alors incohérent. Il est alors possible de récupérer de cette situation grâce aux nombreux outils disponibles dans l'architecture Jini (systèmes de baux, transactions distribuées).

Pour améliorer la robustesse d'un système diffus réparti et pour pallier les fautes, les services de découverte et les services de baux sont souvent mis à contribution. Ainsi, les services s'annoncent périodiquement via le service de découverte et ainsi un service défaillant arrêtera de s'annoncer. Ainsi une imprimante diffusera sa présence toutes les trente secondes sur un réseau local en faisant une annonce à diffusion multiple. Si un nouvel appareil se connecte au réseau, il recevra l'annonce de l'imprimante et pourra l'utiliser. Si cette imprimante cesse de diffuser sa présence sur le réseau parce qu'elle n'a

plus d'encre, les nouveaux appareils ne sauront même pas qu'elle existe et n'essaieront donc pas de l'utiliser. Par contre, l'appareil qui a été notifié de la présence de l'imprimante précédemment pourra se rendre compte de sa disparition, puisqu'il ne reçoit plus d'annonce de celle-ci, et en rechercher d'autres ou en choisir une dans la liste qu'il se sera construite d'après les annonces reçues des autres imprimantes. De même si aucune imprimante satisfaisante n'est disponible, l'appareil qui a besoin d'une imprimante attendra les annonces des autres imprimantes avant de continuer ses activités. Les services ayant reçu une annonce peuvent gérer l'utilisation et la disponibilité d'un service à l'aide de baux. Ainsi, ils sauront qu'un service n'existe plus s'ils ne reçoivent plus d'annonces de la part de ce service. Les processus devront donc toujours être prêts à perdre et à regagner l'accès à des ressources qui sont assujetties à faire défaut.

1.2.8 Sécurité

Les systèmes diffus font face à de nouveaux problèmes de sécurité. Certains appareils doivent être protégés des autres services ou appareils. Il se peut que certains appareils puissent s'intégrer au système informatique diffus sans qu'on leur accorde une confiance totale. Il se peut aussi que l'on veuille garder secrète l'existence de certain services et non se contenter uniquement de bloquer leur utilisation par des appareils étrangers. Par exemple, l'appareil d'un visiteur ne doit pas avoir tous les accès aux services d'une maison. De nouveaux modèles de sécurité doivent être élaborés pour faire face à ce nouveau contexte d'utilisation. La confiance entre les services est un point important dans un système diffus. Les appareils ne se connaissent pas d'avance et les échanges sont nombreux. La sécurité des données transférées lors des communications entre les services et les appareils peut être garantie par l'utilisation de la cryptographie. Il faut bien délimiter le Rubicon sémantique pour prendre les bonnes décisions comme lors de l'échange d'une clef publique pour établir la confiance entre deux services. Les appareils qui n'ont pas assez de ressources pour travailler avec des systèmes de clefs asymétriques peuvent utiliser des clefs symétriques. Mais ils peuvent toujours subir une attaque qui leur fera gaspiller leurs batteries et les rendra hors service. Un service malsain pourrait continuellement

demander l'état d'un appareil à batteries ou bien demander de faire une tâche coûteuse en énergie, ce qui aura comme conséquence de réduire considérablement la longévité des batteries. L'accès à la vie privée des usagers entraîne le contrôle des accès fondé sur l'identité. En effet, un système malveillant pourrait savoir les faits et gestes d'un usager. Une solution est de consentir aux appareils des baux de durée limitée au lieu d'établir des listes de contrôles [3], [5]. L'accès peut aussi être accordé grâce à l'identification, mais sans divulguer l'identité directement. On peut réaliser cela en établissant un lien direct comme une connexion à infrarouge de courte portée entre le nouvel appareil et une composante du système. Plusieurs techniques sont utilisées pour déterminer la présence physique d'un appareil et l'identifier dans un certain périmètre [21], [22]. Ainsi, l'appareil est identifié et on lui accorde les privilèges d'utiliser les ressources du système tout en gardant son identité secrète. Cette action est supervisée par un individu qui peut juger du niveau de confiance que l'appareil lui apporte. C'est une autre décision qui appartient au Rubicon sémantique. Un autre aspect à considérer est la position géographique. Les usagers, au lieu du système, peuvent eux-mêmes établir leurs positions physiques pour préserver leur vie privée. Le système ne devrait pas savoir qui est présent, mais plutôt où il est situé par rapport au système.

Les pare-feu actuels ne sont pas applicables car on veut partager des ressources internes avec des visiteurs provenant de l'externe. En conséquence, la sécurité dans les environnements diffus pose des défis originaux.

1.3 L'informatique dans les centres de logements pour personnes âgées

Pour démontrer les avantages et les applications de l'intégration invisible de l'informatique diffuse dans l'environnement, l'article «Using Pervasive Computing to Deliver Elder Care» [15] survole un système informatique qui a été intégré dans un centre pour personnes âgées. Elite Care a développé une maison de retraite pour personnes âgées en perte d'autonomie pour mettre en application la technologie Care™. L'acronyme Care signifie en anglais

« Creating an Autonomy-Risk Equilibrium », ce qui veut dire : la création d'un équilibre entre l'autonomie et le risque. Cette technologie améliore la qualité de vie des occupants. Les résidents continuent à vivre normalement sans que le médecin vienne faire ses rondes grâce aux senseurs situés partout dans l'édifice. Le but du développement de cette technologie est la diminution des coûts d'intégration de la technologie, la mise en place d'interfaces usagers adaptés aux personnes âgées et la transparence des appareils. Les senseurs, les récepteurs radio, les ordinateurs et les systèmes électriques se fondent dans l'environnement. L'intégration de la technologie se fait donc de manière transparente à l'usager, il n'a pas l'impression d'être entouré d'appareils informatiques et électriques. Autant leur utilisation que leur aspect esthétique est transparent. Avec cette technologie, les employés sont plus efficaces et ils peuvent concentrer leurs efforts sur les besoins réels des occupants. Ils peuvent savoir où et quand ils doivent intervenir. La qualité de vie des employés a aussi augmenté grâce à l'introduction de cette technologie. On incite les employés à vivre sur le campus, ce qui leur permet de travailler depuis leur appartement qui possède toute l'infrastructure informatique. Avec certains employés vivant sur le campus[4], dans les mêmes résidences que les personnes agées, le temps de réaction à des situations d'urgence est ainsi amélioré. Les résidents possèdent un appareil émetteur permettant leur identification et leur localisation sur le campus. Cet émetteur émet à la fois un signal radio et infrarouge. Le signal infrarouge permet la localisation de l'usager dans les pièces des résidences du campus grâce aux capteurs qui sont situés dans chacune de celle-ci. Le signal radio permet de localiser l'usager lorsqu'il est à l'extérieur des bâtiments avec une précision de quatre-vingt-dix pieds. Cet émetteur peut aussi émettre des alarmes. Plusieurs senseurs de poids sont dissimulés dans les lits et dans les toilettes pour pouvoir surveiller les activités et les habitudes de sommeil des résidents. Chaque appartement possède un ordinateur à écran tactile avec une interface usager simple adaptée aux personnes âgées. Cet ordinateur est connecté au réseau. Plusieurs applications y sont disponible : des applications pour entrer en contact avec sa famille et ses amis (par courriel, vidéoconférence en utilisant une caméra web), pour écrire des documents (traitement de

[4] Le campus comporte une superficie de six acres où sont situés les résidences. Jusqu'à quinze personnes peuvent vivre dans une résidence.

texte), etc. Chaque résident possède aussi une base de données personnalisée contenant ses signes vitaux et ses déplacements. Cette base de données est utilisée pour poser des diagnostics sur les patients et pour évaluer l'efficacité des employés. Elle peut aussi servir à encourager les rencontres entre résidents en leur permettant de partager leurs intérêts. La plupart des résidents ne voient pas cette base de données comme une invasion de leur vie privée et acceptent d'y participer. Cette base de données devrait être accessible à distance prochainement.

Le système informatique développé chez Elite Care semble se fondre très bien à l'environnement. Le système semble centralisé et aucune interaction spontanée n'y semble présente. Notre projet de maîtrise s'en rapproche beaucoup en terme de catégories de services offerts à l'usager. Toutefois notre but n'est pas de fournir immédiatement ces services, mais plutôt de concevoir, implanter et valider une infrastructure différente, qui servira de base à un système d'information diffus. Il diffère de celui-ci du point de vue de l'infrastructure. En apparence, les deux projets peuvent sembler très similaires et offrir les mêmes genres de services, mais la différence fondamentale réside dans la manière dont ces services vont coopérer entre eux. Ainsi, notre système n'est pas centralisé. Les composantes s'exécutent indépendamment les unes des autres. Elles utilisent toutefois les autres composantes qui peuvent parfois être disponibles, parfois non disponibles. L'ajout de nouvelles composantes se fait sans configuration de la part des usagers et le système peut prendre avantage des nouvelles composantes de manière transparente. Étant donné que c'est un système distribué, la défaillance d'une composante n'affecte qu'une petite partie du système, mais l'ensemble continue de fonctionner. Dans notre projet l'aspect géo-référencé est aussi présent. Pour la localisation de l'usager ou de services, la précision peut varier de 10 m (avec la technologie Bluetooth) jusqu'à 20 cm dans le cas de l'utilisation de badges d'identification.

CHAPITRE 2

PRÉSENTATION DU PROJET

Ce chapitre présente les éléments essentiels de notre projet de recherche. Le problème consiste à réaliser un système qui permet d'envoyer de l'information à un usager, peu importe où il se trouve et peu importe l'appareil qu'il utilise. Le système doit aussi informer différents usagers. Plus précisément, l'usager notera ses rendez-vous dans l'agenda qui conservera les informations. Au moment opportun, l'agenda enverra à l'usager les informations concernant le rendez-vous en utilisant divers services de l'infrastructure.

Trois scénarios montrent comment sont respectés et appliqués les principes des systèmes informatiques diffus décrits précédemment, à savoir : l'intégration physique, l'inter-opération spontanée, la découverte et l'interaction, l'adaptation, l'intégration, le cadre de programmation, la robustesse et la sécurité. Nous commençons par décrire le laboratoire qui servira de lieu d'expérimentation (§2.1). Ce laboratoire ne change pas d'un scénario d'utilisation à un autre. Nous présentons ensuite les trois scénarios d'utilisation. Chaque scénario intègre un ou plusieurs aspects de l'informatique diffuse. Le premier scénario « Localisation de l'usager » est une adaptation du logiciel de l'agenda face aux allées et venues de l'usager (§2.2). Le deuxième scénario « Migration fluide d'une session en cours » est la réalisation du désir d'un usager de changer d'appareil pour un appareil mieux adapté à ses besoins lorsqu'il travaille dans une application (§2.3). La session en cours n'est pas détruite mais reconstruite le plus fidèlement possible sur le nouvel appareil, là où l'usager était rendu avant la migration. Le troisième scénario « Recherche et avertissement d'un usager » est une utilisation de plusieurs services qui collaborent entre eux pour avertir un usager qu'un rendez-vous approche (§2.4).

2.1 Infrastructure

Pour expérimenter, tester et évaluer des systèmes d'information diffus, il faut mettre en place une infrastructure physique d'une part et une infrastructure logicielle d'autre part. Dans notre cas, l'infrastructure physique est composée de :

- deux détecteurs de mouvements sans-fil X10 (figure 2.1);
- un récepteur radio X10 (figure 2.2);
- une interface X10 pour émettre et recevoir les codes X10 depuis un ordinateur (figure 2.3);
- une imprimante;
- une antenne Bluetooth pour l'imprimante;
- un émetteur de réseau sans-fil (WiFi);
- un PDA (personal digital assistant) à connexion Bluetooth et WiFi (figure 2.4);
- deux ordinateurs;
- une antenne Bluetooth USB pour un ordinateur.

L'infrastructure logicielle à développer et à mettre en place comprendra :

- le logiciel de contrôle et de gestion des codes X10;
- le logiciel de gestion des détecteurs de mouvements;
- le logiciel de l'agenda;
- l'infrastructure Jini;
- les modules Bluetooth pour l'intégration de l'imprimante.

Les différents appareils vont interagir entre eux de diverses manières. Les détecteurs de mouvements vont émettre un signal radio que le récepteur radio X10 captera. Ce dernier transformera les signaux reçus des détecteurs de mouvements en signaux X10 sur le réseau électrique. L'interface X10 capte les signaux X10 qui proviennent du réseau électrique et les transforme pour être utilisés par son interface série qui est branché à un ordinateur. L'ordinateur qui utilise l'interface X10 peut donc, grâce à l'interface, à la fois lire les

signaux X10 qui proviennent du réseau électrique et aussi émettre des codes X10 sur le réseau électrique. L'émetteur de réseau sans-fil sert à intégrer le PDA au réseau local. L'antenne Bluetooth sur l'imprimante permet à l'ordinateur équipé aussi d'une antenne Bluetooth et au PDA de communiquer avec elle. L'ordinateur et le PDA peuvent aussi communiquer directement entre eux grâce à leurs antennes Bluetooth.

Le travail à effectuer consiste à développer les logiciels de contrôle et de gestion des détecteurs de mouvements X10 et des appareils électriques X10 qui sont intégrés au système. Il faut développer l'agenda électronique qui est paramétrique pour pouvoir migrer d'un système à un autre, le serveur http pour le téléchargement instantané du code nécessaire au système Jini et les modules des réseaux Bluetooth et sans-fil pour les imprimantes et les PDAs. Il va falloir déployer une configuration de base de Jini pour établir le système dans lequel les composantes travailleront. Les appareils informatiques et électriques qui sont utilisés pour réaliser les scénarios sont intégrés de manière très simple dans le laboratoire. Le laboratoire consiste en une pièce fermée avec une porte d'entrée. Dans cette pièce, un ordinateur personnel muni d'un écran, d'un clavier et d'une souris, y est allumé (figure 1). Le logiciel client de l'agenda s'y exécute en mode d'attente. Une imprimante avec une connexion Bluetooth y est présente. Un autre ordinateur personnel muni d'un écran, d'un clavier et d'une souris est disposé à l'extérieur de la pièce avec le logiciel client de l'agenda. L'usager est muni d'un PDA qui possède une connexion Bluetooth et WiFi. Un logiciel client de l'agenda s'exécute aussi en mode attente sur le PDA. Le laboratoire dispose d'un réseau informatique (LAN avec un point d'accès sans-fil) et d'un réseau électrique. Le réseau électrique est utilisé par les appareils X10 pour transmettre leurs données et sert aussi de source électrique à tous les appareils électroniques et électriques du laboratoire. Le réseau LAN est réalisé par un routeur standard qui donne aussi l'accès à l'Internet si le besoin s'en fait sentir. Une base réceptrice WiFi est aussi connectée au LAN pour permettre l'intégration d'appareils mobiles.

Figure 2.1: Détecteur de mouvement sans-fil

Figure 2.2: Récepteur radio

2.2 Scénario 1 : Localisation de l'usager

Le premier scénario explore l'intégration invisible de l'informatique dans l'environnement. Il conjugue la localisation de l'usager et la configuration automatique de son environnement en fonction de sa présence. Dans ce scénario, l'usager ne se rend pas compte que plusieurs systèmes informatiques sont activés lorsqu'il entre dans la pièce principale du laboratoire par son entrée unique (figure 2.5). Le logiciel client de l'agenda affiche le menu d'ouverture de session à l'écran de l'ordinateur, avec le nom de l'usager déjà inscrit. Il lui suffit alors seulement d'y inscrire son mot de passe pour débuter sa session. Ce scénario prend comme hypothèse que c'est toujours le même usager qui entre dans la

Figure 2.3: Interface X10

Figure 2.4: PDA

pièce et qu'il n'y a qu'un seul usager dans le laboratoire. Cette restriction est due à la manière dont l'algorithme de détection des personnes fonctionne en collaboration avec les détecteurs de mouvements et aussi à la restriction technique qui empêche d'identifier exactement l'occupant des lieux (puce ou émetteur)[5]. Avec un émetteur que l'usager porte sur lui, le laboratoire aurait pu directement identifier l'occupant puisqu'à chaque occupant aurait été assigné un émetteur avec un code d'identification unique. Si l'usager quitte la pièce sans avoir utilisé l'agenda, le menu de démarrage de session est réinitialisé à l'état d'attente, en particulier le nom de l'usager disparaît.

[5] Au moment du développement et des expérimentations, aucune technologie d'identification n'était disponible au laboratoire DOMUS.

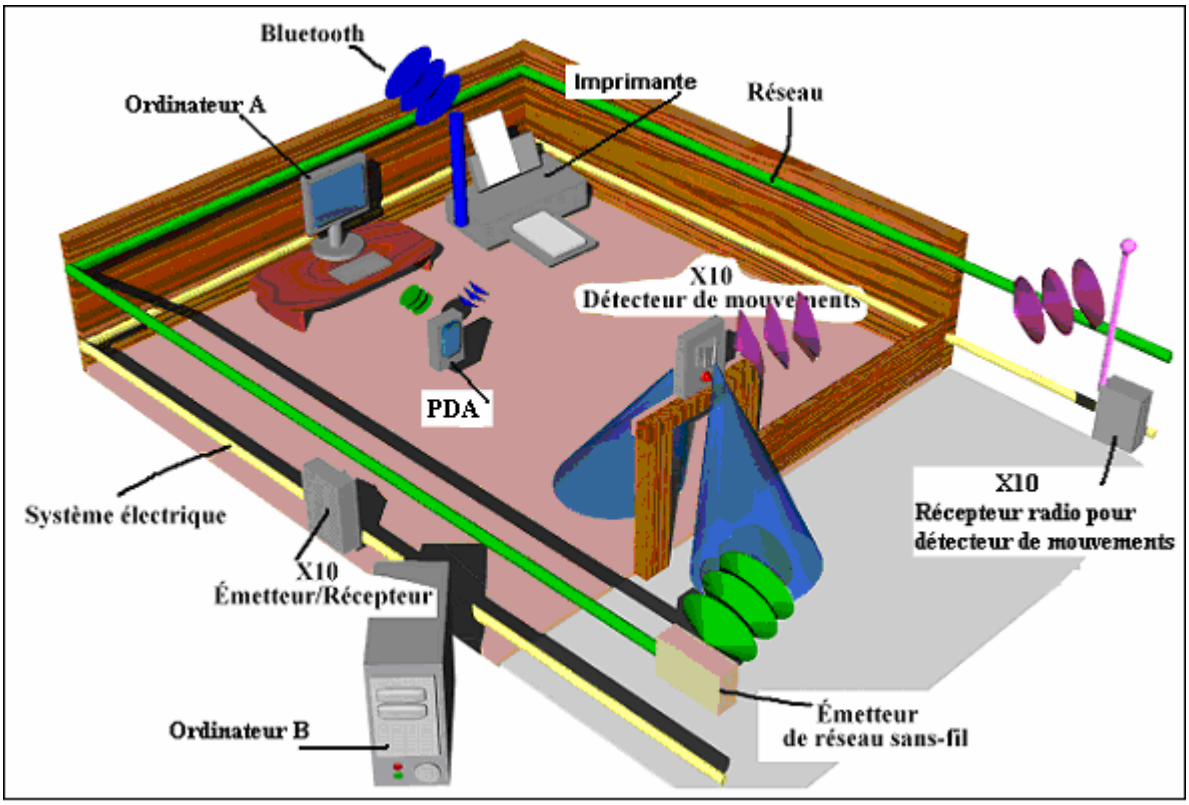

Figure 2.5: Plan du laboratoire. On voit ici le matériel utilisé et sa disposition.

2.3 Scénario 2 : Migration fluide d'une session en cours

Le second scénario explore la mobilité des applications dans l'espace et dans le temps. Pour ce faire, la migration des sessions est utilisée. Dans ce scénario, un usager qui utilise l'agenda désire changer d'appareil pour continuer à travailler. L'usager peut avoir plusieurs raisons qui motivent son désir ou ses besoins : travailler avec un appareil mobile, utiliser un appareil dans une autre pièce, utiliser un appareil disposant d'un meilleur affichage, etc. Deux situations peuvent survenir :

- le transfert de la session se fait directement sur un autre appareil (§2.3.1);
- le transfert de la session se fait par l'intermédiaire du serveur (§2.3.2).

2.3.1 Transfert de session directement sur un autre appareil

Pour étudier la mobilité dans l'espace, la session d'un usager peut être transférée directement sur un autre appareil. Pour ce faire, l'usager peut naviguer pour trouver l'appareil qui convient à ses critères. Lorsque l'appareil désiré est trouvé, il transfère sa session sur celui-ci. La session courante sur l'appareil courant se termine donc automatiquement. Le menu de démarrage de session sur le nouvel appareil s'affiche avec le nom de l'usager déjà inscrit. L'usager doit alors inscrire son mot de passe pour débuter la session. La session est restaurée pour refléter le plus possible l'état où elle était sur l'ancien appareil.

2.3.2 Transfert de la session sur un serveur et récupération à un moment ultérieur

Pour étudier la mobilité dans le temps, la session d'un usager peut être transférée sur un serveur afin de lui permettre de continuer ultérieurement, éventuellement sur un autre appareil. Cette situation peut survenir si aucun appareil n'est disponible, si aucun appareil ne possède les caractéristiques désirées ou si l'usager désire seulement continuer sa session plus tard. Dans l'agenda, l'usager peut naviguer et voir les caractéristiques des autres appareils qui s'offrent à lui. Si les autres appareils disponibles ne l'intéressent pas, l'usager n'a pas d'autre choix que de continuer sa session sur l'appareil courant ou bien de migrer sa session sur le serveur pour être récupérée plus tard depuis n'importe quel appareil. Cette situation peut aussi survenir si les autres appareils qui l'intéressent sont déjà en cours d'utilisation par d'autres usagers. Lorsque l'usager navigue pour trouver d'autres appareils, il peut voir à la fois les caractéristiques et la disponibilité de ces appareils. Pour récupérer une session depuis n'importe lequel des appareils, l'usager ouvre d'abord une session et ensuite choisit de récupérer sa session qui est sauvegardée sur le serveur.

2.4 Scénario 3 : Rappel d'un rendez-vous

Pour communiquer avec un usager, un système d'information diffus doit être capable de le localiser, d'identifier les moyens (appareils) disponibles dans un environnement et finalement de lui présenter l'information. C'est un problème réparti extrêmement complexe. C'est pourquoi le troisième scénario se concentre sur le rappel de rendez-vous pour explorer cette problématique. Ainsi, au moment opportun, le système essaie d'avertir un usager qu'un rendez-vous approche. La procédure d'avertissement se déroule en deux temps. La première étape consiste à émettre un rappel bien avant le rendez-vous (quelques jours). La deuxième étape consiste à prévenir l'usager un peu avant le rendez-vous (quelques heures). Lorsque l'usager reçoit un message de rappel, il entre son mot de passe et peut lire le message. En faisant cela, il confirme au système la réception du message et l'agenda sait que l'usager a été effectivement prévenu de l'approche d'un rendez-vous. Le système cesse alors de tenter de prévenir l'usager de l'approche du rendez-vous. Ce scénario implique la collaboration de plusieurs services entre eux pour réussir à localiser et transmettre un message à l'usager.

CHAPITRE 3

LES TECHNOLOGIES UTILISÉES

Dans notre prototype, plusieurs technologies hétérogènes travaillent ensemble pour réaliser un système informatique diffus. Une caractéristique regroupe toutes ces technologies : le coût. En effet, le coût des appareils et des technologies utilisés est relativement faible ou bien a chuté au cours des dernières années. Dans ce chapitre, nous abordons d'abord les technologies matérielles (§3.1) et ensuite les technologies logicielles utilisées par le prototype (§3.2).

3.1 Les technologies matérielles

Dans cette section sont présentées les composantes matérielles utilisées pour réaliser le projet. Ces composantes ne peuvent travailler directement ensemble pour réaliser les tâches voulues, mais elles pourront le faire une fois intégrées à l'architecture Domus. La figure 2.5 en présente les éléments principaux.

3.1.1 La technologie X10

La technologie X10 repose sur un protocole de communication qui utilise le réseautage électrique déjà en place dans une maison ou un bâtiment. Ce protocole permet aux appareils électriques compatibles X10 de communiquer entre eux. On doit affecter à la main à chaque appareil une adresse parmi les 256 adresses disponibles. Les adresses sont déterminées par une combinaison du code de maison (*house code*) et du code de l'appareil (*unit code*). Les codes de maison vont de la lettre 'A' jusqu'à la lettre 'P', ce qui donne 16 valeurs possibles. Les codes de l'appareil vont du chiffre 1 jusqu'au chiffre 16, ce qui

donne aussi 16 valeurs. Certains appareils ne font que recevoir ou émettre des données, tandis que d'autres appareils peuvent à la fois émettre et recevoir. X10 utilise les cycles électrique pour se synchroniser et envoyer des codes qui se disperseront dans tout le système électrique. X10 réussi à envoyer 2 bits d'information par cycle, mais, par convention, X10 transmet le signal d'un bit dans le premier demi cycle suivi de son complément dans le deuxième demi cycle. Par contre X10 transmet 1 bit de donnée par demi cycle pour les codes de début. Les codes envoyés sur le système électrique peuvent être décomposés comme suit :

Code de début - Code d'adresse - Code de valeur

ou

Code de début - Code d'adresse - Code de fonction

Onze cycles électriques sont utilisés pour transmettre de l'information. Le code de début utilise 2 cycles électriques et chaque cycle contient 2 bits d'information. Le code de début est toujours 1110. Le code d'adresse utilise 4 cycles électriques et chaque cycle contient 1 bit d'information. Les codes de valeur ou de fonction utilisent 5 cycles électriques et contiennent 5 bits d'information. Avec la combinaison des quatres cycles (4 bits) du code d'adresse et des cinq cycles (5 bits) du code de valeur (le cinquième bit du code de valeur ne sert pas lors de l'adressage), on peut obtenir les 256 adresses possibles (4 + 4 = 8bits). Il doit y avoir un silence de 3 cycles électriques entre chaque paire de transmission de 11 cycles électriques. La plupart du temps, les appareils X10 fonctionnent sans problème. Il se peut que dans certaines situations, il y ait des problèmes de communication. Une première source de problèmes survient lorsqu'un autre appareil électrique émet du bruit électrique sur le réseau électrique du bâtiment. Ce sont souvent des appareils à moteur (balayeuse, séchoir à cheveux, malaxeur) et des appareils électroniques très avancés (bloc d'alimentation d'ordinateur portatif, téléviseur à très grand écran, etc.) qui peuvent émettre ce type de bruit indésirable. Pour éliminer ce bruit, on peut brancher l'appareil fautif dans un filtre qui empêche ce bruit électrique de se propager dans tout le réseau électrique. Ces filtres sont des atténuateurs de fréquences. On les branche dans la prise murale et on branche ensuite l'appareil dans celui-ci. Une seconde source de problèmes survient lorsque le transmetteur X10 est situé sur une phase électrique et que le récepteur est situé sur l'autre

phase (figure 3.1 et 3.2). Généralement, le système électrique est réparti sur deux phases et on ne peut pas savoir sur quelle phase une prise

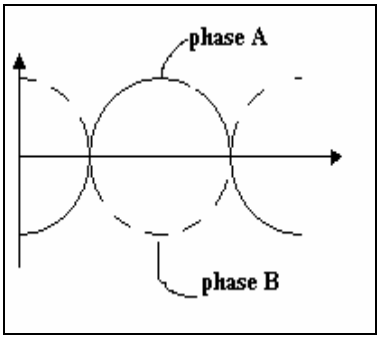

Figure 3.1: Phases électriques.

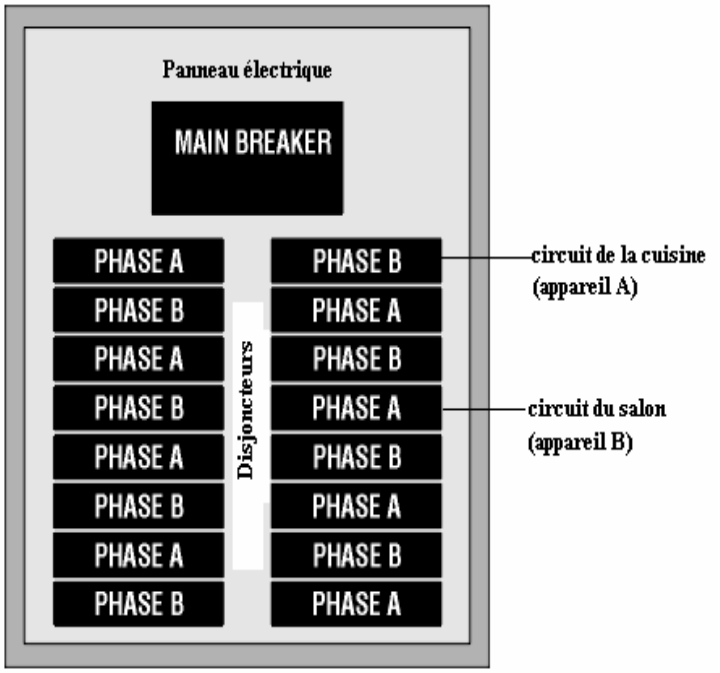

Figure 3.2 : Panneau électrique et les phases associées aux différents circuits.

murale est branchée à moins d'aller voir dans le panneau électrique. Encore une fois, on peut installer un coupleur de phase dans le système électrique pour régler le problème.

X10 a été intégré dans le projet pour son faible coût, pour sa simplicité de configuration, pour son utilisation du système électrique déjà présent et pour la disponibilité d'une multitude d'appareils ayant plus ou moins des fonctions spécifiques à la domotique. Grâce à la technologie X10, on peut aussi allumer ou éteindre des appareils électriques qui ne sont pas conçu a priori pour fonctionner avec X10. X10 rend très flexible l'utilisation d'appareils électriques courants dans le laboratoire et simplifie leur intégration dans l'architecture logicielle du projet. La technologie X10 apporte la possibilité d'utiliser et de contrôler par l'informatique des appareils électriques déjà présents dans toutes les maisons.

Cette intégration de l'informatique avec le système électrique est donc invisible aux yeux des utilisateurs.

3.1.2 Les détecteurs de mouvements

Les détecteurs de mouvements sans-fil apportent un moyen facile et abordable de détecter la présence des usagers dans une pièce et la présence de l'obscurité (ils possèdent aussi une cellule photoélectrique). Les détecteurs de mouvements permettent d'intégrer des attributs physiques (éclairage et présence) au système informatique. Grâce à ceux-ci, le système est encore plus flexible et peut s'adapter à la présence des occupants ou à l'état de l'éclairage de la pièce par rapport à la période dans la journée. Les détecteurs de mouvements résolvent le problème de détection de présence dans la pièce de manière invisible, mais ne résolvent pas le problème d'identification précise de la personne. Le laboratoire utilise deux détecteurs de mouvements sans-fil. Ces détecteurs fonctionnent chacun avec 2 piles AAA et sont disposés dos à dos au-dessus de la porte d'accès du laboratoire. Disposés de cette manière, avec un algorithme informatique, on peut analyser l'état de la pièce en calculant l'ordre d'activation de chacun des détecteurs. Les deux détecteurs de mouvements sont configurés aux adresses E12 (code de maison E et code d'unité 12) et E6 (code de maison E et code d'unité 6) qui deviennent respectivement 4B et 45 en notation hexadécimale. L'interface transforme ces codes d'identification au niveau du port série sous la forme X0*xy* où la variable *x* représente le code de maison et le *y* représente le code d'unité. L'interface transforme aussi les codes d'action sous la forme X1*x*2 pour ACTION ON et X1*x*3 pour ACTION OFF où le *x* équivaut au code de maison. Lorsque le détecteur ressent du mouvement dans son rayon d'opération, il transmet son identité (X0*xy*) suivit du code d'action (X1*x*2 pour ACTION ON et X1*x*3 pour ACTION OFF). Donc lorsque le détecteur de mouvement détecte une présence, il émet sur le réseau électrique le message X04B X142 ensuite, après un certain délai configurable, il émet X04B X143 s'il ne détecte plus de mouvements. Il suffit ensuite de lire ces codes et de les traiter informatiquement pour savoir s'il y a une présence ou non dans la pièce. Les détecteurs de mouvements émettent

aussi des codes pour indiquer le changement d'éclairage dans la pièce. Pour ce faire, ils envoient le code d'identité de la noirceur suivi de l'action ON ou OFF. Le code d'identité pour la noirceur est le code d'unité de l'appareil plus une unité. Donc le premier détecteur de mouvement envoie le code X04C (le code de l'appareil est X04B) suivit de X142 pour indiquer la noirceur (action de la noirceur à ON). Il est à noter que le code d'identification de chacun des détecteurs peut être changé, mais le code d'identité pour la noirceur sera toujours une unité supérieure à l'adresse choisie du détecteur.

3.1.3 La base réceptrice pour signaux radio X10

Comme les détecteurs de mouvements sont sans-fil, une base réceptrice sert à retransmettre sur le réseau électrique les codes d'états qu'ils émettent. Cette base ne fait que retransmettre les codes X10 qu'elle reçoit pour que ceux-ci circulent sur le réseau électrique. Aucune configuration n'est nécessaire. Elle est branchée directement dans une prise de courant. Elle pourrait retransmettre tous les codes X10 provenant de n'importe quels appareils X10 sans-fil.

3.1.4 L'interface LynX-10 PLC

L'appareil LynX-10 PLC sert d'interface entre un ordinateur et le réseau électrique. Il est branché directement dans le réseau électrique à une extrémité et l'autre extrémité est branchée dans un port série d'un ordinateur. Un ordinateur peut ainsi lire et écrire des codes X10 sur le réseau en ayant recours à de la programmation sur le port série. Cet appareil permet de résoudre une partie du problème de l'hétérogénéité des appareils électriques. On peut intégrer des appareils électriques et informatiques et les faire travailler ensemble grâce à des logiciels qui masqueront les couches de bas niveaux impliquées dans la collaboration. Cela permet de contrôler beaucoup plus facilement l'environnement physique dans lequel évoluent les utilisateurs. On peut aussi recueillir des informations

importantes qui rendront le système informatique diffus plus complet à l'aide de senseurs fonctionnant avec X10. Seule l'imagination est un facteur de limitation. Grâce à plusieurs appareils électriques, on peut diagnostiquer plusieurs choses d'une manière peu coûteuse. On peut diagnostiquer les lumières brûlées la nuit en les allumant en alternance et en comparant les informations recueillis par les détecteurs de noirceur par exemple.

3.1.5 Routeur

Pour réaliser un réseau informatique, il faut un routeur. Le routeur permet d'avoir accès à la fois à l'Internet et au réseau local. Il assigne automatiquement des adresses IP aux appareils informatiques qui lui en font la demande. Il permet aux appareils informatiques de communiquer entre eux avec le protocole Ethernet pour les LAN (*local area network*). Le routeur utilisé est un router standard à 4 ports de la compagnie Linksys. Il répond à la norme IEEE 802.3 et IEEE 802.3u.

3.1.6 La base de réception sans-fil

La base de réception sans-fil Lucent AP-500 est connectée au routeur. Elle donne accès au LAN aux appareils avec connexion réseau sans-fil (WiFi). Cela permet d'intégrer les appareils mobiles sans-fil au système. Les appareils mobiles obtiennent leur adresse IP grâce au serveur DHCP du routeur. La base de réception répond aux normes 802.11b.

3.1.7 Les ordinateurs personnels

Pour réaliser le scénario, au moins deux ordinateurs personnels sont nécessaires. Le premier ordinateur personnel (Ordinateur A) est situé dans la pièce principale du laboratoire. C'est celui que l'usager utilise. Le système d'exploitation de cet ordinateur est Windows XP. Java y est installé pour exécuter les applications du système. Cet ordinateur

est un AMD Athlon 1600+ XP avec 512 megs de ram. Il est muni d'un écran couleur, d'une souris, d'un clavier, d'une carte réseau et d'une antenne Bluetooth qui se branche dans le port USB. Le second ordinateur (Ordinateur B) est moins puissant, mais tout aussi efficace. Il utilise Linux Red Hat 9 comme système d'exploitation. Java y est aussi installé pour exécuter les applications du système. Sur cet ordinateur s'exécuteront surtout les logiciels serveurs. L'interface LynX-10 PLC y est branchée dans le port série COM2. C'est donc lui qui récupère et émet les messages X10. Linux a été choisi comme système d'exploitation pour montrer que l'hétérogénéité des systèmes d'exploitation n'est pas un facteur limitatif et qu'avec l'utilisation de Java, on peut réaliser le système de manière peu coûteuse, fiable et robuste. Cet ordinateur est un Pentium 2 à 266 Mhz avec 128 megs de RAM. Cet ordinateur est situé à l'extérieur de la pièce principale du laboratoire. Bien qu'il soit vieux, il est à la hauteur de la tâche qu'il doit réaliser. Son coût matériel est réduit au minimum et le coût du système d'exploitation et des logiciels est nul. Il est muni d'un écran couleur, d'une souris, d'un clavier et d'une carte réseau.

3.1.8 L'assistant personnel iPAQ (PDA)

Un assistant personnel iPAQ H4150 sert d'appareil mobile. Cet ordinateur de poche est muni du système d'exploitation Windows Pocket PC 2003. Il possède une connexion WiFi (WLAN 802.11b) et une connexion Bluetooth intégrée. Le iPAQ a accès au réseau LAN grâce à la base réceptrice. Cet appareil apporte la mobilité tout en gardant un accès au réseau. Il peut exécuter des programmes Java (J2ME). Il possède un affichage couleur de très bonne qualité pour un appareil de cette taille.

3.1.9 L'imprimante sans-fil Bluetooth

Une imprimante est dotée d'un adaptateur Bluetooth HP bt1300 dans la pièce principale. Elle peut être utilisée par n'importe quel appareil possédant une antenne Bluetooth et le

logiciel approprié. C'est à dire qu'un ordinateur ou un PDA peuvent l'utiliser pour imprimer si, à la fois, le logiciel a les capacités d'imprimer et que le système d'exploitation de l'appareil peut aussi utiliser une imprimante. Ce service d'imprimante sans-fil est donc disponible pour les appareils qui sont à proximité (10 mètres) de l'imprimante.

3.1.10 L'antenne Bluetooth branché à un ordinateur

Une antenne Bluetooth est branchée à l'ordinateur A. Cette antenne est vendue avec l'adaptateur Bluetooth pour l'imprimante. Elle se branche dans la prise USB de l'ordinateur qui est dans la pièce principale. Cette antenne permet d'utiliser les autres services Bluetooth qui sont à proximité.

3.2 Les technologies logicielles

Cette section présente les technologies logicielles utilisées pour programmer et réaliser le prototype. On y présente un aperçu de chaque technologie utilisée : Java, RMI, Jini, J2ME, Serveur HTTP, Phoenix, Reggie et Madison.

3.2.1 J2SE (Java 2 Platform, Standard Edition)

Pour réaliser l'architecture principale du projet, Java a été retenu comme langage de base. Java est un langage de programmation multi-plateforme. C'est-à-dire qu'un programme compilé en Java peut être exécuté sur plusieurs plateformes qui ne sont pas compatibles. Les machines virtuelles Java s'occupent d'uniformiser les différences entre les plateformes et d'interpréter le code Java pour qu'il s'exécute correctement sur chaque plateforme. Il existe plusieurs versions de Java car la technologie du langage a été étendue pour fonctionner sur plusieurs types d'appareils et aussi pour répondre à plusieurs besoins spécifiques. Il existe la version standard du langage (J2SE) pour les ordinateurs possédant

des systèmes d'exploitation normaux comme Windows XP, Linux, Unix, etc. Cette version de Java n'est pas restreinte au niveau matériel, ni logiciel. Il existe aussi la version pour les petits appareils comme les PDA et les téléphones cellulaires (J2ME). Cette version doit composer avec les restrictions matérielles imposées par ces appareils. Il y a aussi une distinction à faire entre J2SE et J2EE. J2EE est une plateforme pour développer et déployer des composantes de services multi-tiers qui utilisent le langage Java (J2SE). Une application multi-tiers Java est composée d'une application cliente qui est exécutée sur un ordinateur client, d'une application EJB (Enterprise Java Beans) qui est exécutée sur un ordinateur serveur J2EE et d'une base de données qui est exécutée sur un autre ordinateur serveur. La plateforme J2EE sert surtout aux applications d'entreprises. Le serveur J2EE fournit un environnement pour exécuter et contenir plusieurs EJB à la fois. Il suffit que la bonne machine virtuelle soit disponible pour l'environnement cible. Avec ce langage, on peut aussi développer des interfaces usager d'une manière très simple fonctionnant sur toutes les plates-formes. Java permet de développer toutes les applications nécessaires pour réaliser le projet à un faible coût.

Plusieurs outils et bibliothèques en Java sont déjà disponibles pour la création d'applications. Cela facilite grandement la tâche lorsqu'il s'agit de créer des interfaces usager, de communiquer avec des protocoles réseaux, de travailler avec XML, etc. Le diagramme conceptuel de la figure 3.1 illustre l'intégration des technologies Java à l'intérieur de la plate-forme J2SE. Les API (application programming interface) de Java définissent la manière dont les applications doivent utiliser les classes des bibliothèques J2SE compilées. Les bibliothèques font parties de la plate-forme J2SE. Les interfaces J2SE peuvent être catégorisées comme suit :

- le noyau (« *core java* ») : Le noyau se compose des fonctionnalités essentielles à la création d'applications qui ont besoin d'accès à des bases de données, de sécurité, d'opérations de réseautique et d'entrée/sortie. Ces technologies sont plus axées sur la partie qui est en relation avec le système d'exploitation et les composantes de bas niveau de l'ordinateur.

- les interfaces usager (« *desktop java* ») : Le groupe des interfaces usager (« user interface toolkit ») comprend les technologies nécessaires à la réalisation d'interfaces usager. Ces technologies sont plus axées sur la partie présentation et interaction du logiciel avec l'utilisateur.
- la machine virtuelle : La machine virtuelle Java est responsable de faire le lien entre le matériel, le système d'exploitation, le code compilé Java et la sécurité de la plate-forme. Une machine virtuelle doit exister pour chaque combinaison de matériel et de système d'exploitation que l'on veut utiliser. Les machines virtuelles peuvent provenir de SUN ou d'autres compagnies qui implémentent leur propre machine virtuelle qui sera compatible avec tous les API de la version de J2SE sélectionnée.

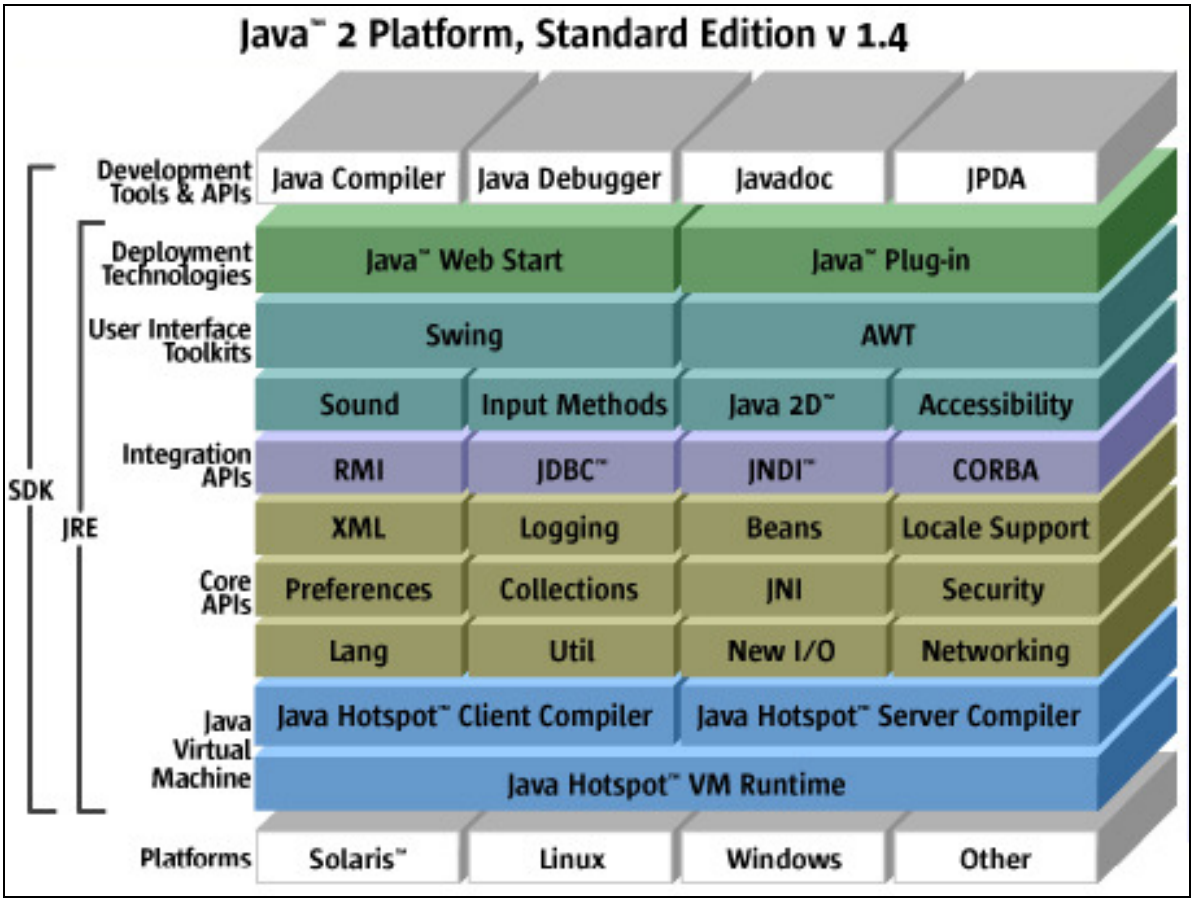

Figure 3.3: Les API disponibles dans l'édition standard de Java.[6]

[6] Illustration provenant du site web http://java.sun.com/j2se/overview.html

3.2.2 J2ME

Le prototype nécessite l'utilisation de J2ME (Java 2 Micro Edition) pour l'exécution du Java sur des appareils aux ressources matérielles restreintes. La plate-forme Java 2, micro-édition (J2ME) est la plate-forme Java pour les petits appareils et produits comme certains appareils sans-fil, les téléphones cellulaires, les ordinateurs de poche et de nombreux appareils électroniques intégrés. L'architecture J2ME définit des configurations, des profils et des paquetages optionnels, pour construire un environnement d'exécution Java qui rencontre les spécifications de plusieurs appareils (figure 3.2). Chaque combinaison est adaptée en fonction de la mémoire, de la puissance de traitement et des capacités des entrées-sorties de chaque appareil.

Les configurations sont composées d'une machine virtuelle et d'un ensemble minimal de classes. Elles permettent de définir des fonctions minimales pour un ensemble d'appareils ayant des caractéristiques similaires. Présentement, seulement deux profils existent : le CDC (Connected Device Configuration) et le CLDC (Connected Limited Device Configuration). La configuration CLDC est la plus petite configuration disponible. Elle a été conçue pour des appareils possédant des connexions intermittentes à un réseau, un processeur assez lent et une mémoire limitée. Ce sont des appareils comme les téléphones cellulaires, les pagettes bidirectionnelles et les PDA bas de gamme. Ces appareils possèdent généralement des processeurs de 16 ou 32 bits. Ils doivent avoir au moins de 128KB à 512KB de mémoire RAM pour la plate-forme Java. La configuration CDC a été conçue pour les appareils qui possèdent des processeurs plus rapides, qui ont une plus grande bande passante sur un réseau et qui ont plus de mémoire. Ces appareils ont généralement des processeurs de 32 bits et possèdent au moins 2MB de mémoire RAM. Dans le prototype, le iPAQ utilise cette configuration.

Dans le but de fournir un environnement d'exécution complet pour un appareil spécifique, une configuration doit être combinée avec un ou des profils qui définissent mieux le cycle de vie d'une application, les interfaces graphiques propres à l'appareil et certaines

propriétés spécifiques. Par exemple, une application qui s'exécute sur un PDA va utiliser la configuration CDC avec le profil *Personal Profile* pour implanter l'interface usager et le profil *Foundation Profile* pour permettre à l'application de communiquer sur le réseau.

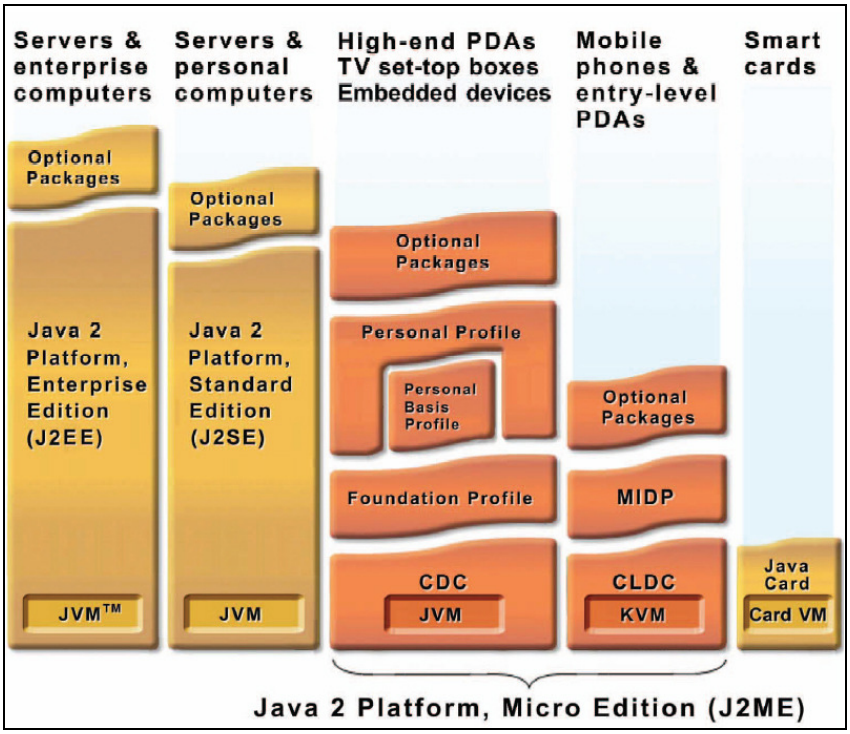

Figure 3.4: Survol de J2ME. [7]

Le profil « *Mobile Information Device Profile* » (MIDP) a été conçu pour les téléphones cellulaires et les PDA bas de gamme. Ce profil contient les interfaces usager, les connexions réseaux et le stockage local de données. Combiné à CLDC, il fournit un environnement d'exécution Java complet.

La configuration CDC est fait de manière à ce que l'on ajoute par couches certains profils et paquetages jusqu'à l'obtention de l'environnement d'exécution désiré. Le profil «*Foundation*» est le profil de plus bas niveau. Il implante les connexions réseaux, sans fournir d'interface usager. Le profil « *Personal Profile* » vise les appareils qui peuvent avoir des interfaces usager évoluées comme les PDA haut de gamme. Il contient le « *Java*

[7] Illustration provenant du site web http://java.sun.com/j2me/j2me-ds.pdf

Abstract Windows Toolkit » au complet et plusieurs autres fonctionnalités. Le paquetage optionnel RMI pour J2ME fournit les mêmes outils RMI pour la communication réseau que le J2SE. Dans le prototype, le iPAQ utilisé supporte la configuration CDC et utilise quelques profils pour pouvoir s'intégrer à l'architecture du système. Le iPAQ utilise le profil « *Personal Profile* » et le « *Foundation Profile* ».

3.2.3 RMI

Une des technologies très utilisée dans le projet est RMI (Remote Method Invocation). RMI est une technologie Java qui permet au programmeur de faire abstraction des couches de communication lorsqu'il développe une application distribuée. Ceci facilite grandement son travail. Les méthodes d'une application Java peuvent être appelées par une autre application Java qui s'exécute dans une autre machine virtuelle. Cette autre machine virtuelle peut exister soi sur le même ordinateur ou sur un autre ordinateur sur un réseau. Le RMI utilise la « *marshalisation* » pour envoyer et recevoir les paramètres. La « *marshalisation* » survient lorsqu'il y a un transfert d'objets Java d'une machine virtuelle vers une autre. RMI va transformer l'objet en une série d'octets qui sont transférés vers l'autre machine virtuelle. L'objet Java est ensuite reconstruit dans l'autre machine virtuelle à partir de ces octets soit par valeur, soit par référence. Lorsqu'un objet désire utiliser un autre objet à distance, il doit avoir une référence à cet objet à distance. Cette référence est acquise par l'entremise du serveur qui s'exécute sur le même ordinateur que l'objet à distance. Ce serveur est un service d'enregistrement de noms des objets et s'appelle RmiRegistry. Il sert aux objets qui désirent être accessibles à distance à s'enregistrer sous un nom pour qu'ils puissent être découverts et référencés. Les objets utilisent alors ce service pour retrouver les objets enregistrés par leur nom d'enregistrement. L'objet demandeur obtient alors une référence à son objet distant et peut interagir avec lui par la «*marshalisation* ».

3.2.4 JINI

Jini est le nom d'une plateforme pour réaliser des systèmes répartis. L'infrastructure de Jini repose sur Java et le RMI pour le transfert d'objets entre les applications. Jini ne supporte pas directement les interactions avec J2ME et on doit utiliser la technologie Madison pour intégrer le IPAQ à la fédération Jini (§3.2.8). Grâce à Java, les applications Jini peuvent migrer et s'exécuter sur une grande variété de plates-formes.

Jini défini plusieurs protocoles qui servent au bon fonctionnement de ces applications réparties. Jini offre plusieurs services pour réaliser des systèmes répartis : réseautage spontané, services de découverte, transactions réparties, système de baux. Jini permet à des services de se rendre disponibles aux autres services pour réaliser leurs tâches. Les services, grâce à l'architecture de Jini, peuvent s'adapter aux changements de configuration du réseau, à l'arrivée de nouveaux services et à la disparition de services. Cette architecture met en place des outils pour permettre aux services de se publier ou de trouver des services qui leur conviennent.

Les caractéristiques des réseaux Jini sont nombreuses. Les réseaux Jini peuvent s'autorétablir et s'autoconfigurer grâce au système de baux qui gère le renouvellement de l'adhésion des services à une fédération. Tous les services Jini sont organisés sous forme de fédérations. Ces fédérations sont des ensembles qui regroupent un ou plusieurs services. Chacun des services doit au moins appartenir à une fédération, mais il peut aussi appartenir à plusieurs fédérations. Jini fournit plusieurs services de base pour aider à la recherche de services dans les fédérations, se joindre aux fédérations, effectuer des transactions distribuées, etc. Finalement, on peut aussi réaliser le réseautage spontané grâce à Jini. L'architecture s'adapte automatiquement aux changements de configuration du réseau.

Pour réaliser un système réparti Jini fonctionnel, on doit lancer plusieurs services de base. Les services nécessaires sont un serveur HTTP ou tout type de serveur qui peut utiliser d'autres protocoles de communication pour la distribution du code mobile (§3.2.5), un

service de localisation et d'enregistrement (§3.2.7) et un service d'activation RMID (RMI Daemon §3.2.6). Ces services de base sont le cœur de l'infrastructure Jini et seront utilisés par les services qui se joindront à la fédération.

3.2.5 Le serveur HTTP

Un serveur HTTP est nécessaire au bon fonctionnement de la fédération Jini. Pour qu'un service puisse être disponible dans la fédération, il doit exporter une composante proxy ou un objet sérialisé au service de localisation. L'objet proxy contient l'adresse pour récupérer le code sur le réseau. Cette composante sera alors envoyée au client pour que celui-ci puisse utiliser le service. Pour que le service soit utilisé par le client, celui-ci a besoin à la fois du code et des données. Pour rendre ce code disponible, on utilise un serveur HTTP qui doit être accessible par le client. Le code transféré qui provient du serveur HTTP est la définition et l'implantation des classes à exécuter. Ce code peut être directement emmagasinées sous la forme d'un fichier compilé Java (.class) ou bien sous la forme d'un fichier archivé (fichier .jar) qui peut contenir plusieurs fichiers compilés ainsi que des ressources comme des images. Le client récupère la partie statique du code sur le serveur HTTP et les données (la valeur des variables pour recréer les objets Java) sont transférées directement entre le service et le client en utilisant la « *marshalisation* » des données. Cette technique rend les fédérations Jini très flexibles. Chaque service doit donc rendre disponible le code de son proxy par un serveur HTTP. Les services peuvent se partager ces serveurs HTTP, il n'est pas nécessaire d'avoir un serveur HTTP pour chaque service. Une fois les données et le code de la classe téléchargés, le client peut recréer l'objet Java pour l'utiliser.

3.2.6 PHOENIX

Phoenix est un service propre à Jini. Phœnix est une implémentation du système d'activation de RMI. Il permet d'établir la communication RMI et d'avoir des services de type activables. La caractéristique activable des services provient des protocoles de Jini et est indépendante de RMI. Pour qu'un service soit activable, il doit respecter plusieurs caractéristiques. Un service de type activable s'enregistre auprès du système d'activation et ensuite se termine. Lorsqu'un client a besoin de ce service, le système d'activation tente de ramener ce service en état fonctionnel pour qu'il soit utilisable par le client. Ce système permet d'économiser la mémoire de l'ordinateur dans lequel le service s'exécute. L'utilisation de services activables est transparente pour le client. Ce système est aussi persistant, c'est à dire que si on arrête Phœnix pour une raison ou une autre (ordinateur instable ou seulement besoin de redémarrer), tous les services enregistrés dans Phœnix seront encore disponibles lorsque celui-ci est redémarré. Phœnix possède toutes les caractéristiques de RMID[8] (*Remote Method Invocation Daemon*), mais en plus il possède des caractéristiques de sécurité configurables.

3.2.7 REGGIE

Reggie est un service de découverte propre à Jini. Il supporte le réseautage spontané. Il sert à enregistrer et à localiser les services. Ce service permet aux autres services de s'enregistrer dans une fédération Jini et de trouver d'autres services dans la fédération. Cette composante contient les proxy des services qui désirent être disponibles dans la fédération. Ce proxy est la partie visible du service et c'est avec cette partie que le client interagit avec le service. Reggie est interrogé par les clients qui cherchent des services spécifiques. Dans le prototype, ce service est persistant et activable grâce au système Phœnix.

[8] RMID est une implantation de base du système d'activation de RMI qui est disponible dans le J2SE.

3.2.8 MADISON

Pour pallier au fait que Jini n'est pas directement supporté par J2ME, on utilise l'architecture Madison. Madison est une implémentation conforme de la spécification de l'architecture Surrogate [1] pour Jini et de la spécification IP Interconnect [14] pour Jini. L'architecture Surrogate a pour but d'intégrer à l'architecture Jini les appareils qui ne supportent pas complètement la technologie Jini. Il s'agit souvent d'appareils qui ne supportent que J2ME. La spécification IP Interconnect définit le protocole de découverte entre l'appareil et le conteneur d'applications Surrogate et la façon dont le conteneur Surrogate télécharge, exécute et maintien actif l'application qui représente l'appareil (l'application surrogate). Dans le prototype, le logiciel qui s'exécute sur le iPAQ utilise J2ME et la configuration CDC avec le « *Personnal Profile* ». Cette architecture permet d'utiliser Madison pour intégrer le iPAQ à l'architecture Jini. Madison agit comme un conteneur d'application Surrogate pour faire le lien entre les appareils et la fédération Jini.

CHAPITRE 4

ARCHITECTURE LOGICIELLE

Pour réaliser le système de messagerie diffuse, trois groupes de logiciels doivent être mis en place et exécutés. Le premier groupe est tout ce qui se rapporte à Jini et son bon fonctionnement. Le deuxième groupe se rapporte aux serveurs qui effectueront des tâches qui sont invisibles à l'usager. Finalement le troisième groupe de logiciels contient ce qui se rapporte aux interactions avec l'usager, c'est-à-dire les logiciels qui ont des interfaces usagers et qui attendent des commandes de l'usager. Dans ce chapitre, nous présentons chaque groupe avec leurs particularités de configuration et d'exécution. La figure 4.1 montre les applications qui sont impliquées dans le système de messagerie diffuse. Toutes les applications appartenant à la couche de l'infrastructure Domus ont été développées pour ce projet. Les applications appartenant à la couche Jini sont déjà existantes et elles sont utilisées pour créer une infrastructure Jini qui va servir à la réalisation du système de messagerie diffuse.

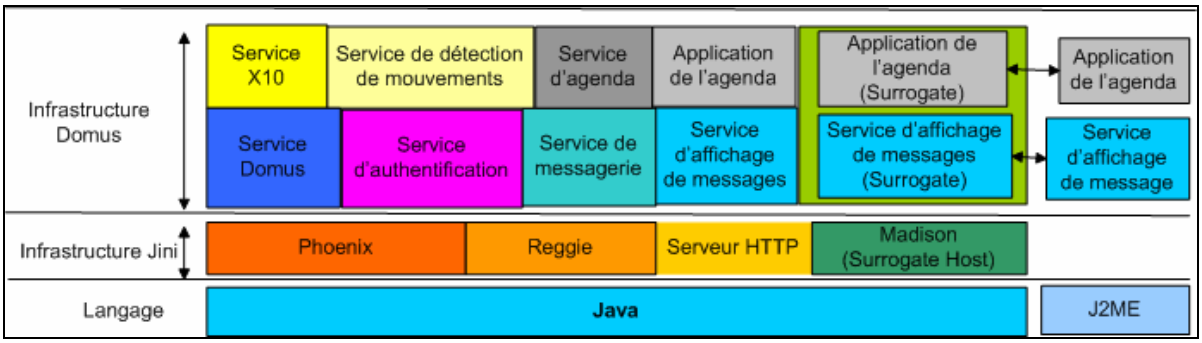

Figure 4.1 : Modèle stratifié du système de messagerie diffuse.

4.1 Jini

La base de l'architecture du projet repose sur Jini. Nous n'avons pas à modifier les composantes Jini mais nous devons choisir comment elles s'exécuteront. Pour avoir un système réparti Jini, quelques services doivent être exécutés et mis en place. Ces services sont un serveur HTTP, le service de persistance Phœnix, le service d'enregistrement et de localisation de Reggie et Madison pour l'intégration des applications J2ME. Finalement les fédérations servent à regrouper logiquement les services disponibles dans l'architecture Domus. Une fois ces services en place, les applications développées pour réaliser les scénarios peuvent interagir entres-elles.

4.1.1 Le serveur HTTP des services Jini

RMI est utilisé comme mécanisme de base dans le système pour la communication interprocessus. Comme les appareils qui se joindront au système diffus ne sont pas connus à priori, ils ne possèdent pas nécessairement les classes nécessaires. Pour rendre les classes Java disponibles sur le réseau local, on utilise un serveur HTTP. Un serveur HTTP de base écrit en Java est disponible dans les outils fournis avec Jini. Ce serveur n'est pas un service Jini, mais simplement un serveur HTTP dont l'adresse IP est déterminée par l'ordinateur qui l'exécute. Le serveur utilise le port 8080 et est exécuté par l'ordinateur B[9]. Le port 8080 est utilisé pour éviter un conflit car les serveurs HTTP standards utilisent généralement le port 80. Plus tard, un autre serveur HTTP utilisera le port 8081 sur ce même ordinateur. Le serveur HTTP est le premier programme à être lancé pour réaliser une architecture Jini. Il dessert les fichiers « jars » qui contiennent le code mobile de Reggie, ce qui permettra aux autres services de trouver et accéder les services disponibles. Le serveur HTTP est lancé avec la commande suivante :

java -jar ..\jini2_0_002/lib/tools.jar -dir ..\jini2_0_002\lib -port 8080 -trees –verbose

[9] L'ordinateur B, le moins puissant, est situé à l'extérieur de la pièce principale.

4.1.2 Phoenix

Phœnix est un programme Java utilisé par Reggie pour rendre ce dernier activable et persistant. Phœnix doit être configuré à l'aide de plusieurs fichiers et lancé avec des paramètres. Les fichiers de configuration sont *« phoenix.config »* et *« phoenix.policy »*. Le fichier *« phoenix.config »* contient le répertoire utilisé pour sauvegarder les informations relatives à la persistance. Le fichier *« phoenix.policy »* contient les paramètres de sécurité à appliquer lors de l'exécution de Phoenix. Phoenix s'exécute sur l'ordinateur B et doit être lancé avant Reggie.

Phoenix est lancé par la ligne de commande suivante :

```
java    -Djava.security.policy=phoenix\phoenix.policy \                          (1)
        -Djava.rmi.server.codebase=http://%computername%:8080/phoenix-dl.jar \   (2)
        -jar ..\jini2_0_002\lib\phoenix.jar \                                    (3)
        phoenix/phoenix.config                                                   (4)
```

(1) Le premier paramètre spécifie à la machine virtuelle Java le fichier de sécurité à utiliser.
(2) Le deuxième paramètre spécifie à la machine virtuelle Java la localisation et le nom du fichier contenant le code mobile à télécharger.
(3) Le troisième paramètre spécifie à la machine virtuelle Java le fichier à exécuter.
(4) Le quatrième paramètre indique à l'application Phoenix le fichier de configuration à utiliser.

4.1.3 Reggie

Reggie est l'application qui permet à une fédération Jini de fournir un répertoire pour les services qui désirent offrir leurs services aux autres usagers. Reggie est lancé en collaboration avec Phœnix. Reggie devient alors un service « activable » et persistant. Reggie est lancé à la suite de Phoenix sur l'ordinateur B. Il sera en mode attente jusqu'à ce qu'un service faisant appel à lui veuille l'utiliser. Reggie a aussi comme mandat de créer

les fédérations Domus. Ces fédérations sont créées lors du lancement de Reggie, elles sont spécifiées en paramètres. Il est à noter qu'un service qui désire faire partie d'une fédération ne doit pas nécessairement s'exécuter sur le même ordinateur que celui où réside le Reggie qui a créé cette fédération. Dans une architecture Jini, l'ordinateur sur lequel s'exécute Reggie ou tout autre service n'a pas d'importance une fois que le service est fonctionnel. D'autre Reggies peuvent aussi s'exécuter ultérieurement sur le même ordinateur ou sur différents ordinateurs pour créer d'autres fédérations.

Reggie est lancé par la ligne de commande suivante :

java *-jar ../jini2_0_002\lib\start.jar * **(1)**
reggie\start-activatable-reggie.config **(2)**

(1) Ce paramètre indique le fichier .jar contenant l'exécutable Java (Reggie).
(2) Ce paramètre indique à Reggie le fichier de configuration à utiliser.

Le fichier *«start-activatable-reggie.config»* contient toutes les informations relatives à l'exécution de Reggie. Entre autre, il contient l'emplacement du répertoire à utiliser pour la persistance et l'emplacement du code mobile à télécharger.

4.1.4 Madison

Lorsque l'application agenda s'exécute sur le iPAQ (J2ME), celle-ci contacte Madison. Madison exécute alors l'application surrogate Java qui se joindra à la fédération Jini à la place du iPAQ. Cette application fait donc le lien entre l'agenda sur le iPAQ et la fédération Jini. L'application agenda et son application surrogate qui est exécutée par Madison utilisent un protocole spécifique qui vérifie que chacune des parties de l'application est bien active. Lorsqu'une des applications se termine à cause d'une erreur ou parce que l'appareil sans-fil est hors de portée du récepteur sans-fil, l'autre partie s'en rend compte et se termine sans erreur. Le conteneur Madison retire alors l'application et

attend une nouvelle demande de la part du iPAQ. L'application agenda communique avec Madison et avec son représentant dans la fédération Jini par des datagrammes. Un sous-projet de Domus a été créé en utilisant la version de Jini 1.2 pour faire la démonstration du système Madison car ce dernier utilise Jini 1.2 pour fonctionner et le projet principal a été développé en utilisant la version Jini 2.0. Par contre, la même version de Java a été utilisée dans chacun des cas (Java 1.4).

4.1.5 Les fédérations de Domus

Jini permet de réaliser des fédérations virtuelles. Ces fédérations virtuelles permettent de regrouper des services selon des caractéristiques choisies. Un service peut se joindre à une ou plusieurs fédérations s'il le désire. Dans le projet, les fédérations représentent des pièces physiques. Cette représentation est utile puisque certains services peuvent ne pas être intéressés par un service, qui répond aux critères recherchés, car il n'est pas situé dans la bonne pièce. Les fédérations agissent comme des groupes. Les groupes peuvent être disjoints ou avoir une intersection. Le contenu des fédérations change continuellement à chaque fois qu'un nouveau service apparaît ou quitte le système.

Dans le système de messagerie diffus, la fédération DOMUS représente toute la maison ou le laboratoire et la fédération "ChambreA" représente la pièce principale (figure 4.2). Un service peut s'inscrire dans l'une ou l'autre des fédérations ou les deux s'il est sensé offrir son service à l'une ou l'autre des fédérations. La localisation physique d'un service peut avoir une importance primordiale pour l'usager. Un écran tactile situé sur un mur a une utilité si l'occupant est dans la même pièce, mais il n'est d'aucune utilité s'il est situé sur le même mur mais du coté opposé dans l'autre pièce connexe. Les services qui ont besoin de ce type d'appareils doivent donc faire la distinction lorsqu'un service est présent dans l'architecture Domus et aussi lorsqu'un service est présent dans certaines fédérations et non dans d'autres. Dans d'autres cas, certains services n'ont pas d'impact physique (un service d'entreposage de données par exemple) avec les utilisateurs alors ils peuvent s'inscrire dans

d'autres fédérations, mais il est important de conserver une certaine discipline logique pour que le système évolue bien. Dans une maison plus grande, avec une multitude de services, les fédérations auront une plus grande importance. Les fédérations pourront regrouper les services par étages, par pièces, par type de service (électrique, X10, chauffage, éclairage etc.). Dans notre architecture, les deux fédérations sont créées par le service Reggie qui s'exécute sur l'ordinateur B.

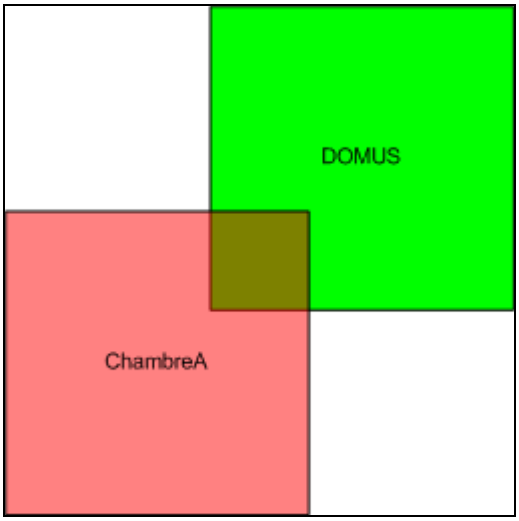

Figure 4.2: Fédérations du laboratoire.

4.2 Descriptions des fichiers de configuration

Tous les services qui s'intègrent dans le système Domus utilisent des fichiers de configuration. Ces fichiers servent à configurer les logiciels pour augmenter leur flexibilité dans la fédération et pour mieux personnaliser le système s'ils sont déployés dans un environnement différent. En définissant la configuration dans les fichiers, on évite de recompiler les logiciels pour les déployer dans différents types d'environnements, peu importe la plateforme. Les fichiers de configuration prennent la forme d'un fichier texte avec le nom d'une propriété suivie d'une valeur. La figure 4.3 énumère les propriétés

d'ordre général communes à toutes les applications. Cet exemple montre le fichier de configuration du service d'agenda.

```
(1) Domus.federation.nom=Domus
(2) Domus.service.nomDuJar=AgendaServer-dl.jar
(3) Domus.federation.a.joindre_0=Domus
(4) Domus.primaire.federation_0=ChambreA
(5) Domus.primaire.seulement=oui
(6) Domus.primaire.federation.unique=oui
(7) Domus.secondaire.federation_0
(8) Domus.secondaire.federation.unique=non
```

Figure 4.3: Exemple des paramètres de configuration du service d'agenda.

Voici les explications des paramètres et les valeurs possibles qu'ils peuvent prendre :

(1) Domus.federation.nom : Ce paramètre spécifie le nom de la fédération principale de l'architecture Domus. Il est obligatoire car cette fédération est à l'origine des services de base de l'architecture. Il peut prendre comme valeur n'importe quel nom que l'on donne à la fédération Domus.

(2) Domus.service.nomDuJar : Ce paramètre indique au service le nom du fichier jar qui est nécessaire au bon fonctionnement du service. Ce fichier (AgendaServer-dl.jar) contient les classes à être distribuées par le serveur HTTP de Domus[10]. Ces classes seront téléchargées par les clients de ce service. Ce fichier contient entre autre l'interface de l'objet Proxy du service qui sera enregistré auprès de Reggie. Le service d'Agenda tentera d'envoyer le fichier AgendaServer-dl.jar au service Domus pour que ce dernier rende disponible via son serveur HTTP le code mobile de l'application du service agenda. Le service d'agenda s'enregistre ensuite auprès de Reggie en donnant comme adresse du code mobile celle fournie par le service Domus. Il ne faut pas confondre les deux serveurs HTTP qui sont intégrés dans l'architecture Domus. Le premier serveur HTTP étant celui

[10] Il est à noter que ce n'est pas le serveur HTTP pour la configuration de base pour Jini mais bien un autre serveur HTTP fourni comme service dans la fédération Domus. Ce serveur sera présenté dans la section suivante.

qui est nécessaire à l'architecture Jini (§4.1.1) et le second serveur HTTP est celui qui est fourni par un service tout usage appelé Domus pour la distribution du code mobile provenant des services faisant partie de l'architecture Domus (§4.3.1).

(3) Domus.federation.a.joindre : Ce paramètre indique au service une fédération qu'il doit joindre pour offrir ses services. Étant donné la logique du service, il peut joindre une ou plusieurs fédérations. On ajoutera alors le paramètre au fichier de configuration en prenant soin d'incrémenter le nombre de une unité pour chacune des fédérations supplémentaires à joindre.

(4) Domus.primaire.federation_0 : Les groupes primaires et secondaires servent à définir les fédérations où chercher les services. Les fédérations secondaires peuvent servir de fédération de secours à choisir en second lieu si l'application a besoin d'une alternative. Ce paramètre sert à spécifier le ou les groupes (fédération Jini) primaires où chercher un service désiré. Ces services sont ceux dont le service qui s'inscrit a besoin pour réussir sa tâche. Dans l'exemple de la figure 4.4, le service d'agenda recherche le service de messagerie dans la fédération Domus (cf. fig. 4.4). Le type de service à chercher n'est pas spécifié dans le fichier de configuration mais dans le code de l'application. Les services trouvés dans le où les groupes primaires auront préséance sur les services trouvés dans les groupes secondaires. Pour ajouter des groupes supplémentaires, on incrémente le nombre de 1 tout en répétant le paramètre et en spécifiant la fédération additionnelle. Le service d'agenda peut se servir de cette fonctionnalité pour raffiner ses recherches ou bien si elle connaît d'avance l'emplacement d'un service qu'elle préfère utiliser.

(5) Domus.primaire.seulement : Ce paramètre indique si le service recherché doit seulement faire partie du ou des groupes primaires spécifiés. La valeur peut être **oui** ou **non.** Si le service recherché ne fait pas partie de la fédération primaire, l'application ne trouvera pas ce service, même s'il est présent dans une autre fédération.

(6) Domus.primaire.federation.unique : Ce paramètre indique si le service spécifié doit être unique. C'est-à-dire que si plusieurs services sont trouvés faisant partie du ou des groupes primaires, seulement un seul service sera retenu. La valeur peut être **oui** ou **non.**

(7) Domus.secondaire.federation_0 : Ce paramètre sert à spécifier le ou les groupes (fédération Jini) secondaires où chercher le service désiré. Pour ajouter des groupes

supplémentaires, on incrémente le nombre de 1 tout en répétant le paramètre et en spécifiant la fédération additionnelle.

(8) Domus.secondaire.federation.unique : Ce paramètre indique si le service spécifié doit être unique. C'est-à-dire que si plusieurs services sont trouvés faisant partie du ou des groupes secondaires, seulement un seul service sera retenu. La valeur peut être **oui** ou **non.**

La figure 4.4 montre un exemple d'utilisation du fichier de configuration.

```
Domus.primaire.seulement=non
Domus.primaire.federation.unique=non
Domus.primaire.federation _0=Domus
```

Figure 4.4 : Utilisation du fichier de configuration 'messagerie.config'

Ce fichier (messagerie.config) est utilisé par le service d'agenda pour trouver un service de messagerie. Il recherche le service de messagerie dans la fédération « Domus » (Domus.primaire.federation_0=Domus) et peut utiliser plusieurs services de messagerie s'il y en a plusieurs (Domus.primaire.federation.unique=non). Il n'est pas non plus obligé d'utiliser des services qui sont découverts dans la fédération primaire (Domus.primaire.seulement=non). Le type de service n'est pas spécifié dans le fichier de configuration mais dans l'objet Java qui utilise ce fichier.

4.3 Les services

Cette section présente les services mis en oeuvre dans le système de messagerie diffuse : les services généraux de Domus, le service d'authentification, le service de messagerie, le service X10, le service de détection de mouvements, le service d'agenda et le service d'affichage de message. Les serveurs sont invisibles pour l'usager. Ils travaillent ensemble mais sans nécessiter d'intervention de l'usager ou presque. Ils n'ont donc pas besoin d'interfaces graphiques pour les utiliser ou gérer les options.

4.3.1 Les services généraux de Domus

Le service Domus est utile à tous les autres services de l'architecture Domus. Il est principalement composé de deux parties : une partie serveur HTTP et une partie qui est service dans l'architecture Domus (appellé le service d'information). La partie serveur HTTP permet le transfert de fichiers pour la distribution du code mobile. Sa fonctionnalité est la même que le serveur HTTP nécessaire au bon fonctionnement de Jini sauf qu'il sert à distribuer le code mobile des applications qui ont été développées pour le projet Domus. Il s'agit d'un serveur HTTP standard, écrit en Java fourni comme outil avec l'ensemble des services base de Jini. Il s'exécute sur le port 8081 pour ne pas entrer en conflit avec l'autre serveur HTTP de Jini (§4.1.1). Cette fonctionnalité simplifie le déploiement et l'exécution des autres services qui feront appel à cette fonctionnalité. Ils n'auront pas à fournir eux-mêmes de serveur HTTP pour distribuer leurs classes.

Le service d'information Domus se joint à la fédération Domus et rend accessible le serveur HTTP aux autres services. Le serveur HTTP devient ainsi accessible en tant que service Jini dans la fédération Domus. Les services désirant bénéficier du serveur HTTP transféreront alors leur code mobile au service d'information Domus qui le rendra disponible sur le LAN via le serveur HTTP. On utilise ce serveur HTTP en plus du serveur HTTP de Jini pour rendre l'architecture de Domus modulable et aussi pour augmenter sa robustesse. Le service d'information de Domus peut donc contrôler l'ajout de nouveaux fichiers qui seront disponibles sur le serveur HTTP. Le service d'information Domus informe aussi le service ayant fait la demande de l'adresse exacte du serveur HTTP. Cette adresse HTTP est nécessaire au service qui utilise le serveur HTTP pour distribuer son code. Le service (le service d'agenda par exemple) qui rend son code disponible par le serveur Domus doit savoir où celui-ci rend les classes disponibles. Les clients de ce service (les clients qui utilisent l'agenda) doivent télécharger le code mobile du service et c'est à ce dernier de fournir cette information et non le service Domus. Cette adresse qui indique où réside le code mobile est nécessaire pour le bon fonctionnement de la couche RMI.

Le service Domus s'inscrit dans la fédération DOMUS puisque c'est un service d'ordre général. L'usager n'est jamais invité à configurer quoi que ce soit sur ce service. Il est totalement invisible à l'usager. Ce service n'est pas essentiel dans l'architecture Domus, mais sans le service Domus, les services ne fournissant pas de serveur HTTP pour rendre leur code disponible ne peuvent fonctionner correctement. Pour augmenter la simplicité du développement, du déploiement et de l'exécution des services, tous les services utilisent Domus pour la distribution de leur code mobile. Le serveur Domus est un service optionnel mais il réduit la quantité de serveurs HTTP qui peuvent être nécessaires dans l'architecture Domus.

Voici l'interface des services offerts par le serveur Domus (figure 4.5) :

```
public interface DomusInfoService  {
(1)     public String getServiceDescription() throws RemoteException;
(2)     public String getClassServerURL() throws RemoteException;
(3)     public boolean putJar(String userName, String password, FilePacket jar) throws
        RemoteException;
}
```

Figure 4.5: Services offert par le service Domus.

(1): Cette méthode retourne une brève description textuelle décrivant le service.

(2): Cette méthode retourne l'adresse URL du serveur HTTP de Domus. Les services ayant recours à ce serveur doivent connaître cette adresse pour pouvoir informer leurs clients de l'emplacement du code mobile à télécharger.

(3): Cette méthode permet aux services de transférer au serveur Domus le fichier jar contenant leur code mobile. Le serveur Domus rendra alors ce fichier disponible via le serveur HTTP.

4.3.2 Le service d'authentification

Le service d'authentification sert à authentifier les usagers et les services qui en ont besoin. Ce service s'enregistre dans la fédération Domus puisqu'il est d'ordre général. Le service Agenda et les clients de la messagerie utilisent beaucoup ce service. Le service d'authentification utilise le service Domus pour distribuer son code mobile. Le service d'authentification est essentiellement une base de données qui contient les noms des usagers et leur mot de passe et qui est accédée comme service dans la fédération Domus par ceux qui en ont de besoin. Un usager peut être une personne ou un service informatique. Voici l'interface des services offerts par le service d'authentification (figure 4.6) :

```
public interface SecurityPortalService {
(1)     public boolean authenticate(UserId_Impl param) throws RemoteException;
}
```

Figure 4.6: Services offerts par le service de sécurité.

(1) Cette méthode retourne la valeur vrai si l'objet *param* reçu en paramètre correspond à une authentification valide. La classe UserId_Impl contient un nom d'usager et un mot de passe. Si le mot de passe ne correspond pas à celui dans la base de données, l'identification est rejetée. La figure 4.7 illustre le code source utilisé pour une demande d'authentification. Pour des fins de développement, le serveur créera un nouveau compte pour toute demande d'un nouvel usager avec son mot de passe lors de la première demande d'identification de celui-ci. Ce service est très minimal. On pourrait très facilement améliorer ce service pour qu'il utilise de la cryptographie et une base de données pour en faire un système 3-tiers très sécuritaire, mais cela dépasse la portée de ce travail.

4.3.3 Le service de messagerie

Le service de messagerie permet à n'importe quel utilisateur d'envoyer un message à un occupant de la maison. Ce système est conçu pour envoyer un ou des messages à un destinataire et, le cas échéant, transmettre un accusé de réception à l'auteur du message. Ce

```
public boolean anthenticate(UserId_Impl param) throws RemoteException {
        if (param == null)
                return false;

        UserId_Impl realUserID = (UserId_Impl) getUserTable().get(param.getUserID());
        if (realUserID == null) {
                addUser(param);
                return true;
        }
        if (realUserID.getPassword().equals(param.getPassword())) {
                return true;
        }
        return false;
    }
```

Figure 4.7 : Code qui vérifie un usager par le service d'authentification.

service permet de gérer les demandes et les confirmations de réceptions de messages. Il gère aussi la validité des messages selon certains critères. Un critère peut être une date d'échéance du message. Une fois cette date dépassée, le service de messagerie retire le message pour ne pas causer d'engorgement du système. Le système de messagerie fait partie de la fédération Domus puisque ce service est conçu pour être utile à n'importe quel système ou usager dans le laboratoire Domus. Les contraintes physiques ne s'appliquent pas à son exécution. Ce système est le meilleur moyen pour tenter de localiser un occupant du laboratoire et de le contacter. Lorsque le service de messagerie reçoit une confirmation de réception de message, il connaît le moment de la réception et le service utilisé. Un autre service pourrait être élaboré pour prendre ces informations et localiser un usager physiquement dans l'appartement. Voici l'interface des services offerts par le serveur Domus (figure 4.8) :

```
public interface MessageFeedbackService  {
(1)     public void notifyMessageDelivered(ServiceID clientThatDelivered, MessageIDImpl theMessage
(2)     public MessageIDImpl deliverMessage(UserId_Impl target, MessageImpl theMessage,
                                    RemoteMessageListener theSource, MessageIDImpl id)
(3)     public MessageIDImpl stopDeliveringMessage(MessageIDImpl theMessage)
}
```

Figure 4.8: Services offerts par le service de messagerie.

(1) Cette méthode est utilisée par les récepteurs de messages. Lorsque le destinataire d'un message réussit à lire le message qui lui est destiné, l'application réceptrice du message informe alors le service de messagerie qu'une livraison de message a été faite avec succès au bon destinataire. Le paramètre ServiceID est un identificateur unique identifiant les services à l'intérieur d'une fédération Jini. Dans ce cas particulier, cet identificateur sert à identifier le destinataire du message qui confirme la réception. Le paramètre MessageIDImpl est un identificateur unique qui sert à identifier le message qui fut correctement livré à un usager.

(2) Cette méthode est utilisée par un service (ou un client) qui désire utiliser le serveur de messagerie. Le paramètre UserId_Impl identifie le destinataire du message. MessageImpl est l'objet qui contient le message à être envoyé. L'objet RemoteMessageListener est l'objet à être informé (le client lui-même ou tout autre entité) lors de la confirmation de réception du message. L'objet MessageIDImpl est l'identificateur unique du message. Il contient des informations et est modifié par le serveur. Le serveur assigne un numéro de message unique pour ne pas qu'il y ait de conflits entre les messages et les identificateurs.

(3) Cette méthode est utilisée par un service qui avait fait une demande de transmission de message et qui ne désire plus que le message soit transmis au destinataire. Il envoie en paramètre l'identificateur unique du message à annuler.

La transmission complète d'un message d'un émetteur au destinataire se passe en six étapes :

Étape 1 : L'émetteur du message localise le service de messagerie et lui fait parvenir sa demande de messagerie en lui transmettant le message et le destinataire.

Étape 2 : Après avoir reçu une demande, le service de messagerie localise tous les clients de réceptions de messages dans toutes les fédérations qu'il connaît.

Étape 3 : Une fois les clients de réception de messages trouvés, le service fait parvenir le message à tous les clients de réception de messages (tout en incluant le destinataire du message).

Sous étape 3.1 : Cette sous étape se réalise de manière propre au client mais devrait respecter l'aspect de confidentialité du message entre l'émetteur et le destinataire. Le client affiche une fenêtre affichant le nom du destinataire. Si le destinataire est présent, il peut entrer son mot de passe et lire le message. Cette action sert de confirmation de la réception du message. Si le destinataire n'est pas présent, la fenêtre reste en attente.

Sous étape 3.2 : Une fois l'authentification par le service d'authentification effectuée, le client de réception de message contacte le service de messagerie pour confirmer la livraison du message au récepteur. Le client de réception de message contacte le service de messagerie seulement si l'authentification a été réalisée avec succès.

Étape 4 : Le service de messagerie reçoit une confirmation de réception de la part d'un récepteur de message.

Étape 5 : Lorsque le service de messagerie reçoit une confirmation, il retire les autres messages qui peuvent être toujours en attente dans les autres clients récepteurs de messages.

Étape 6 : Le service de messagerie contacte l'émetteur du message pour confirmer que le message a été livré et lu par la bonne personne.

Pour assurer un état valide du système, d'autres opérations sont effectuées afin qu'il n'y ait pas d'accumulation de messages non valides dans les clients récepteurs de messages et aussi afin que tout les messages encore valides soit transmis à des clients récepteurs de messages qui viennent de se joindre au système. Voici ces opérations qui assurent un état valide du système :

Opération 1 : À tout moment, l'auteur d'un message peut demander au service de messagerie de retirer un message. Le service de messagerie localise alors les récepteurs de

messages (service d'affichage de messages) et leur demande de retirer le message en question.

Opération 2 : De nouveaux récepteurs de messages peuvent se joindre à la fédération à tout moment. Le serveur leur fera alors parvenir les demandes de livraison de messages en cours pour avoir les meilleures chances de contacter l'usager.

Opération 3 : Des récepteurs de messages peuvent être coupés de la fédération à tout moment. Il se peut donc que certaines demandes ne leur soient pas parvenues. Le service de messagerie leur fait part des messages à retirer s'ils n'en ont pas déjà été informés lorsqu'ils se rejoignent à la fédération.

Voici le schéma de collaboration :

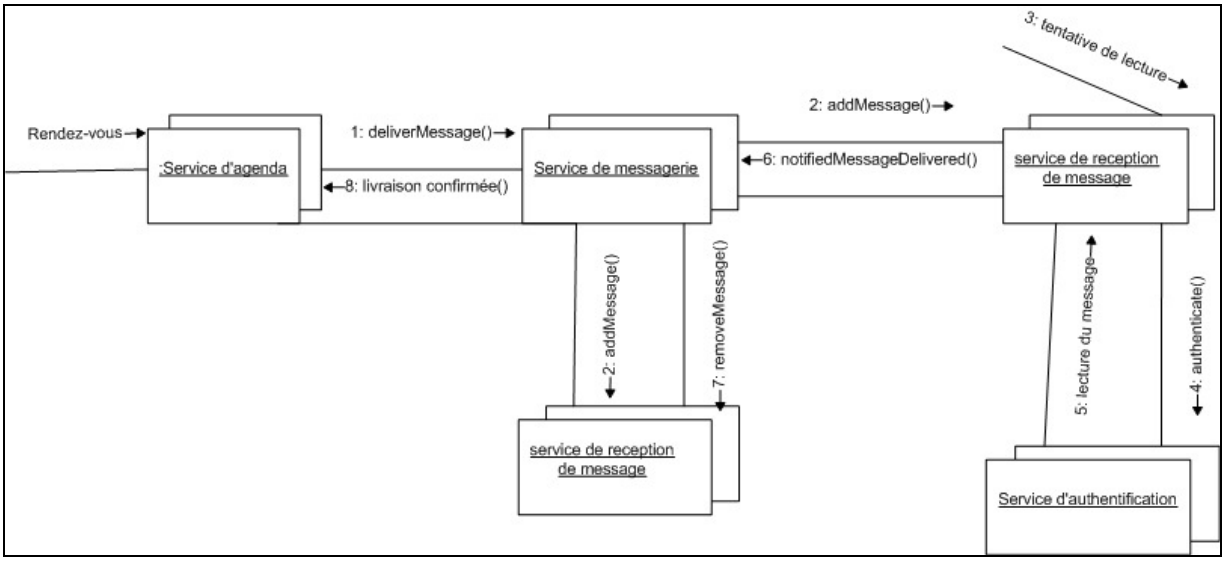

Figure 4.9: Schéma de collaboration d'un envoi de message.

4.3.4 Le service X10

Le service X10 permet aux appareils X10 de s'intégrer dans l'architecture logicielle de Domus. Il fait partie de la fédération Domus car le réseau électrique est partagé par toutes les pièces du laboratoire. Le service sert de canal de transmission des codes X10 provenant du réseau électrique pour les retransmettre aux applications Domus qui sont intéressées par

ces derniers. Ces applications peuvent aussi envoyer des codes X10 sur le réseau électrique en utilisant ce service. Le service X10 est réalisé en deux couches. La première couche touche l'aspect matériel et la deuxième partie est l'aspect logiciel. Le matériel comprend l'interface LynX10-PLC qui est branché dans une prise murale et dans le port série de l'ordinateur B. La partie logicielle comprend un module qui communique avec l'interface LynX10-PLC par le port série de l'ordinateur et un autre module qui intègre les services dans la fédération Jini.

Voici l'interface du service X10 (figure 4.10) :

```
public interface X10Service  {
(1)     public EventRegistration addX10EventListener(String x10Code, X10RemoteEventListener listener)
(2)     public boolean removeX10EventListener(String x10Code, X10RemoteEventListener listener)
(3)     public boolean removeX10EventListener(X10RemoteEventListener listener)
(4)     public boolean writeX10Code(String code)
}
```

Figure 4.10: Services offerts par le service X10.

(1): Cette méthode est utilisée par un autre client (ou service) qui désire recevoir un code X10. Le client connaît d'avance le code qui peut circuler sur le réseau électrique et veut en être informé lorsque celui-ci circule pour le traiter. Le paramètre x10Code est le code X10 et le paramètre « *listener* » est l'objet qui sera notifié de la présence de ce code.

(2): Cette méthode est utilisée pour informer le serveur X10 qu'un certain service (listener) n'est plus intéressé par le code X10 spécifié en paramètre (x10Code).

(3) : Cette méthode est utilisée pour indiquer au serveur X10 qu'un certain client (listener) n'est plus intéressé à recevoir des informations par rapport aux codes X10 circulant sur le réseau électrique.

(4) : Cette méthode permet aux clients (ou autres services) d'écrire des codes X10 (paramètre code) sur le réseau électrique.

Le service X10 utilise une bibliothèque optionnelle Java pour communiquer avec le port série de l'ordinateur sur lequel il s'exécute. Cette bibliothèque est différente pour chacun

des systèmes d'exploitation et cette différence est prise en compte dans le code de Java. Par contre, le même service X10 peut s'exécuter autant dans l'environnement Windows que dans l'environnement Linux à condition que la bibliothèque soit correctement installée sur chacun des systèmes. La figure 4.11 montre le code source qui prend en charge la bibliothèque pour accéder au port série de l'ordinateur.

```
[...]
        loadJavaxCommProperties();
         if(OS_NAME.startsWith("Win")) {
            [...]
            loadWin32ComDllLibrary();
            [...]
            loadWin32DriverLibrary();
        }
[...]
        public void loadWin32ComDllLibrary() throws Exception {
           [...]
            System.loadLibrary("win32com");
           [...]
        }

        public void loadWin32DriverLibrary() {
            String driverName = "com.sun.comm.Win32Driver";
                  [...]
            javax.comm.CommDriver driver =
                              (javax.comm.CommDriver)Class.forName(driverName).newInstance();
          driver.initialize();
          System.out.println("Win32Driver Initialized");
           [...]
         }
```

Figure 4.11 : Extrait du code source pour initialiser la bibliothèque qui donne l'accès au port série.

4.3.5 Le service de détection de mouvements

Le service de détection de mouvements comporte un attribut lié au monde physique dans l'architecture de Domus. C'est un service qui procure des informations sur l'état d'une pièce. Il fait donc partie de la fédération "ChambreA" puisque c'est cette dernière qu'il surveille. La localisation physique a un grand impact sur la disponibilité du service dans l'architecture logicielle. Il ne faut pas induire en erreur d'autres services qui ont besoin de la fonctionnalité d'un détecteur de mouvements dans une pièce donnée. Différents types de

services peuvent aussi être intéressés à plusieurs détecteurs de mouvements localisés dans plusieurs pièces. Dans l'architecture actuelle, c'est à eux de gérer l'appartenance des détecteurs de mouvements à telle où telle fédération.

Le service de détection de mouvements de la chambre A est composé de deux parties. La première partie matérielle consiste en deux détecteurs de mouvements disposés dos à dos au dessus de la porte d'entrée. Cette disposition permet de détecter les allées et venues dans la pièce. Les détecteurs de mouvements communiquent par ondes radio avec la base réceptrice. Celle-ci retransmet les codes X10 sur le réseau électrique du laboratoire. La deuxième partie, qui est logicielle, est composée de deux modules. Le premier module est une partie client qui se sert du serveur X10 afin de recevoir les codes envoyés par les détecteurs de mouvements. Le deuxième module est le service qui est offert à la fédération de la chambre A pour connaître l'état de la pièce. D'après l'ordre de réception et le temps écoulés entre les codes X10 reçus, le service détermine s'il y a une présence ou non dans la pièce. Le service reçoit aussi des informations sur l'état de l'éclairage de la pièce.

La figure 4.12 donne l'interface du service de détection de mouvements. Cette interface comprend les méthodes suivantes :

- **(1) :** Cette méthode donne l'identité de la pièce sur laquelle le service donne des informations.
- **(2) :** Cette méthode donne l'information la plus à jour sur le statut de la pièce.
- **(3) :** Cette méthode donne l'état de noirceur de la pièce.
- **(4) :** Cette méthode permet de s'inscrire auprès du détecteur de mouvement pour recevoir un avertissement lorsque quelqu'un entre dans la pièce.
- **(5) :** Cette méthode permet de s'inscrire auprès du détecteur de mouvement pour recevoir un avertissement lorsque quelqu'un quitte la pièce.
- **(6) :** Cette méthode permet de se retirer de la liste des avertissements des évènements lorsque quelqu'un quitte la pièce.
- **(7) :** Cette méthode permet de se retirer de la liste des avertissements des évènements lorsque quelqu'un entre dans la pièce.

(8) : Cette méthode permet de s'inscrire auprès du détecteur de mouvements pour recevoir tous les types d'évènements.

(9) : Cette méthode permet de se retirer de la liste de tous les évènements de la pièce pour ne plus les recevoir.

(10) : Cette méthode permet de recevoir des événements sur l'état de noirceur de la pièce.

```
public interface DetecteurMouvementService {
(1)     public String getRoomID()
(2)     public boolean isSomeonePresent()
(3)     public boolean isRoomInDarkness()
(4)     public void addRoomEnteredListener(RoomEnteredListener l)
(5)     public void addRoomExitedListener(RoomExitedListener l)
(6)     public void removeRoomExitedListener(RoomExitedListener l)
(7)     public void removeRoomEnteredListener(RoomEnteredListener l)
(8)     public void addRoomStatusChangedListener(RoomStatusChangedListener l)
(9)     public void removeRoomStatusChangedListener(RoomStatusChangedListener l)
(10)    public void addRoomDarknessListener(RoomDarknessListener l)
(11)    public void removeRoomDarknessListener(RoomDarknessListener l)
}
```

Figure 4.12: Services offerts par le service de détection de mouvements.

(11) : Cette méthode permet de ne plus recevoir des événements sur l'état de noirceur de la pièce.

Pour réussir à détecter un usager qui quitte ou qui s'introduit dans la pièce principale du laboratoire, il faut développer un algorithme qui tient compte de l'ordre d'activation des détecteurs de mouvements ainsi que de l'intervalle du temps écoulé entre ces activations. Étant donné l'hypothèse qu'il y a seulement un seul usager en même temps dans le laboratoire, on peut traiter quatre cas principaux et quelques variantes. Pour débuter, on suppose que le laboratoire est vide. Référer à la figure 2.5 pour visualiser l'orientation des détecteurs de mouvements. Le premier cas est une simple activation du détecteur de mouvement situé à l'extérieur de la porte. Ce cas-ci ne modifie pas l'état de la pièce

puisque le détecteur intérieur n'est pas activé. Le deuxième cas est une activation en chaîne du détecteur extérieur suivit du détecteur intérieur dans un intervalle de temps assez court. Dans ce cas, on conclu que l'usager est entré dans la pièce et on peut lancer un événement d'un usager qui entre dans la pièce. Le troisième cas est une simple activation du détecteur interne sans activation du détecteur externe. On peut conclure que l'usager se promène à l'intérieur de la pièce mais ne la quitte pas, il n'y a donc aucun changement de l'état de la pièce. Le quatrième cas est une activation en chaîne du détecteur interne suivit du détecteur externe dans un intervalle de temps assez court. Dans ce cas, on peut conclure que l'usager quitte la pièce et on peut lancer un événement pour indiquer que la pièce est maintenant vide. Les autres variantes possibles sont toutes les autres possibilités d'activation avec un usager déjà présent ou avec une pièce vide. La figure 4.13 illustre la partie de l'algorithme qui détecte le départ de l'usager.

```
 TimeStamp commandTimestampA;  //heure d'activation du détecteur interne
TimeStamp commandTimestampB;  //heure d'activation du détecteur externe
[...]
if (commandTimestampA != null && commandTimestampB != null &&
commandTimestampB.after(commandTimestampA))
{
      if (CommandA.equals(getACTION_ON()) && CommandB.equals(getACTION_ON()) &&
                  commandTimestampB.getTime() - commandTimestampA.getTime() < 10 * 1000)
      {
                  setSomeOnePresent(false);
                  CommandA = null;
                  CommandB = null;
                  commandTimestampA = null;
                  commandTimestampB = null;
      } else if (CommandA.equals(getACTION_OFF()) && CommandB.equals(getACTION_ON()))
      {
                  //FAIT RIEN
      } else if (CommandA.equals(getACTION_ON()) && CommandB.equals(getACTION_OFF()))
      {
                  setSomeOnePresent(true);
                  CommandA = null;
                  CommandB = null;
                  commandTimestampA = null;
                  commandTimestampB = null;
      } //else do nothing
}
```

Figure 4.13 : Extrait de l'algorithme de détection du départ d'un usager de la pièce.

4.3.6 Le service d'agenda

Le service agenda constitue le cœur du système. Il sert de répertoire central pour emmagasiner les rendez-vous des usagers. Le service d'agenda vérifie l'identité des usagers en utilisant le service d'authentification de Domus lorsque ceux-ci l'utilisent. L'usager peut ajouter ses rendez-vous à l'aide du client de l'agenda. Une fois les rendez-vous sauvegardés sur le serveur, ce dernier essaie de contacter les usagers par divers moyens à sa disposition afin de rappeler à l'usager qu'un rendez-vous approche. Le serveur essayera de contacter deux fois un usager. Le premier rappel a lieu longtemps avant le rendez-vous (quelques jours). Le deuxième rappel a lieu quelques heures avant le rendez-vous. Pour avertir l'usager, le serveur agenda utilise le service de messagerie le mieux adapté pour transmettre des messages à des usagers. Lorsqu'un usager confirme un rendez-vous, le service d'agenda en sera averti. Le service d'agenda peut alors se créer une base de données pour savoir quels rendez-vous ont été rappelés à l'usager (1er rappel ou 2^{e} rappel).

Lorsqu'un usager n'a pas été contacté à temps, le service agenda se charge aussi de signaler au service de messagerie de retirer les messages périmés. Le service agenda sert aussi de passerelle lorsque des clients de l'agenda désirent transférer leur session en cours sur un autre poste informatique (voir la section 4.4.2). La figure 4.14 présente l'interface que le service d'agenda offre aux clients :

```
public interface AgendaServerService  {
(1)     public boolean addMigrationDescription(UserId_Impl param,
SessionDescriptionImpl param2)
(2)     public boolean addRendezVous(UserId_Impl param, RendezVousImpl param2)
(3)     public SessionDescriptionImpl getMigrationDescription(UserId_Impl param)
(4)     public RendezVousImpl[] getRendezVous(UserId_Impl param)
(5)     public void removeRendezVous(UserId_Impl param, RendezVousImpl param2)
(6)     public boolean logonUser(UserId_Impl param)
}
```

Figure 4.14: Services offerts par le service d'agenda.

(1) : Cette méthode permet à un client agenda de transférer une session en cours sur le serveur agenda. Cette méthode prend un objet décrivant l'usager et un objet décrivant la session comme paramètres.

(2) : Cette méthode permet d'ajouter un rendez-vous à la liste des rendez-vous de l'usager.

(3) : Cette méthode permet d'obtenir une session en cours pour l'usager.

(4) : Cette méthode permet d'obtenir la liste des rendez-vous pour l'usager.

(5) : Cette méthode permet de supprimer un rendez-vous pour l'usager.

(6) : Cette méthode permet à un client de l'agenda d'authentifier un usager qui désire utiliser le service d'agenda.

4.3.7 Le service d'affichage de messages

Les services d'affichage de messages sont utilisés par le service de messagerie pour afficher des messages provenant de sources diverses. Ils s'occupent de l'authentification de l'usager pour que le message ne soit uniquement lu que par le destinataire. Ces services peuvent être autonomes ou intégrés dans un autre logiciel. Dans le système, l'application agenda intègre à la fois le service d'agenda et le service d'affichage de messages. Le service d'affichage n'est affiché que lorsqu'il y a des messages en attente d'être lus. C'est-à-dire qu'aucune interface graphique n'est présentée à l'usager s'il n'y a pas de message en attente d'être lu. Le service d'affichage de messages s'enregistre dans une fédération pour être trouvé par le service de messagerie. On le considère aussi comme client puisque l'usager s'en sert pour interagir avec le service de messagerie et du service d'authentification.

Il existe deux méthodes d'authentification. La première utilise le service d'authentification Domus. La seconde est indépendante du service d'authentification, c'est-à-dire que le service de messagerie envoie au service d'affichage de messages à la fois le message, le nom du destinataire et le mot de passe qui reste caché. Le service d'affichage de messages

compare alors le mot de passe fourni par le service de messagerie avec le mot de passe fourni par l'usager dans la fenêtre de saisie. Cette seconde méthode est plus robuste que la première en cas de défaillance du service d'authentification de Domus. Dans notre cas, on utilise toujours le service d'authentification.

La figure 4.15 présente l'interface offerte par le service d'affichage de messagerie :

(1) : Cette méthode permet au service de messagerie d'ajouter un message à transmettre à un usager. Le paramètre ServiceID est l'identifiant unique du service de messagerie. Le client a besoin de cet identifiant pour retrouver le bon service de messagerie lorsqu'il tentera d'avertir celui-ci de la réception du message par le destinataire. Le paramètre MessageImpl est l'objet contenant le message et le destinataire.

(2) : Cette méthode permet au service de messagerie de demander au service d'affichage de messagerie de retirer le message qui correspond à l'identificateur unique de message.

```
public interface MessageFeedbackClientService {
(1)     addMessage(ServiceID serverServiceID, MessageImpl theMessage)
(2)     removeMessage(MessageIDImpl theMessage)
}
```

Figure 4.15: Services offerts par un service d'affichage de messages.

4.4 L'application agenda

Le logiciel de l'agenda électronique est la seule application utilisée directement par l'usager. À première vue, cet agenda se comporte comme un agenda informatique standard. On peut y ajouter des rendez-vous (figure 4.16) avec la description (date, heure, description du rendez-vous, lieu), on peut supprimer des rendez-vous au besoin (figure 4.17) et on peut naviguer à travers la liste de rendez-vous (figure 4.18). Par contre, cet agenda diffère des autres agendas sur les aspects suivants :

- Il supporte plusieurs usagers.
- Les rendez-vous sont sauvegardés sur un serveur.
- Il est possible de changer d'ordinateur en cours d'utilisation
- L'agenda est utilisé en conjonction avec le service d'agenda qui tente d'avertir l'usager à l'approche d'un rendez-vous.
- L'agenda intègre aussi un service d'affichage de messagerie. Lorsque l'application s'exécute, elle enregistre à la fois le service d'affichage et l'application agenda dans l'architecture de Domus.

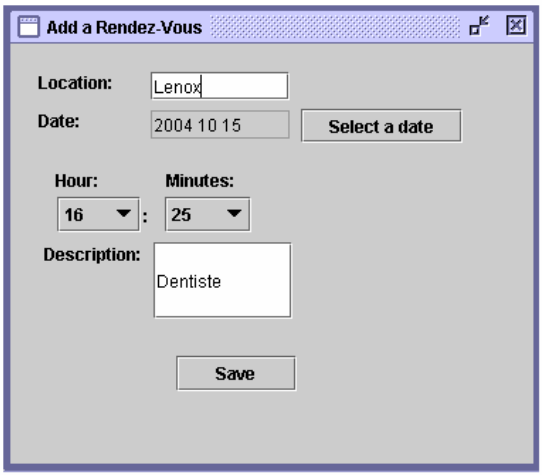

Figure 4.16: Ajout d'un rendez-vous.

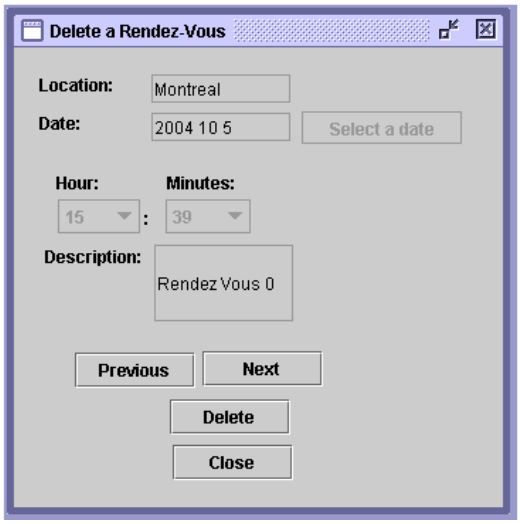

Figure 4.17: Suppression d'un rendez-vous.

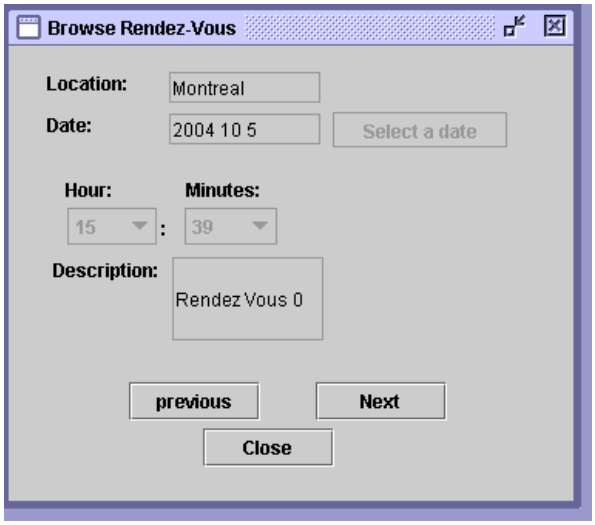

Figure 4.18: Consultation des rendez-vous.

La figure 4.19 montre l'interface des services offerts par l'application de l'agenda. Cet interface comprend quatre méthodes :

```
public interface AgendaClientService {
(1)     getComputerDescription()
(2)     getUserInfo()
(3)     isAvailable()
(4)     pushMigrationSession(SessionDescriptionImpl param2)
}
```

Figure 4.19: Services offerts par un client agenda.

(1) : Cette méthode permet à un client agenda d'avoir la description de l'ordinateur sur lequel s'exécute l'application.

(2) : Cette méthode permet de savoir qui utilise l'application au moment de la demande.

(3) : Cette méthode permet de savoir si l'application est disponible à l'utilisation (pour une migration de session).

(4) : Cette méthode permet à un autre client de l'agenda de télécharger une session en cours pour la poursuivre sur ce poste de travail. Le paramètre contient l'information nécessaire pour rebâtir une session à l'endroit où l'usager était rendu. L'information nécessaire pour recréer la session contient une généralisation de l'application. Cette généralisation est une manière simple de paramétrer la position des fenêtres ouvertes et leur contenu. Chaque fenêtre est représentée par un objet sérialisable qui contient les informations pour recréer la fenêtre et son contenu. La figure 4.20 montre la classe Java utilisée pour recréer la fenêtre et la figure 4.21 montre la classe Java utilisée pour recréer le contenu.

L'application agenda qui s'exécute sur un iPAQ utilise le conteneur surrogate Madison. Le surrogate de l'agenda du iPAQ permet de se joindre à la fédération Domus. La partie qui se joint à la fédération Domus est totalement compatible avec Jini et l'architecture Domus ne fait donc pas la différence entre celle-ci et les autres agendas. C'est donc au surrogate

de l'agenda et de l'agenda sur le iPAQ de transférer tous les paramètres nécessaires au bon fonctionnement de l'agenda.

Une session consiste, du point de vue d'un usager, à inscrire son nom d'usager et son mot de passe. Une fois ces informations validées, l'application offre les fonctionnalités d'un agenda. L'usager peut ensuite utiliser les menus et les multiples fenêtres pour ajouter, supprimer et consulter des rendez-vous. Une session est constituée alors du contenu des fenêtres, de leur position et de l'état de l'application (quelles fenêtres sont visibles).

```
public class SessionRebuilderImpl implements Serializable
{       private boolean EstVisible=false;
        private Point Point= null;
        private Dimension dimension= null;
        private RestoreParamImpl Param= null;
        public SessionRebuilderImpl() {}
        public boolean EstVisible() {return EstVisible; }
        public Point getPosition() {return point;}
        public Dimension getDimension() {return dimension; }
        public RestoreParamImpl getSessionParam() {return param;}
        public void setDimension(Dimension dimension) {this.dimension = dimension;  }
        public void setVisible(boolean b) { EstVisible = b;}
        public void setPosition(Point point) {   Point = point; }
        public void setRestoreParam(RestoreParamImpl param) { Param = param; }
}
```

Figure 4.20: Représentation d'une fenêtre.

```
public class RestoreParamImpl implements Serializable
{       private static final long serialVersionUID = -7657220650172857832L;
        private Hashtable param= null;
        public RestoreParamImpl() {}
        public void addParam(String key, Object Param) {getParam().put(key, Param);}
        public Object getParam(String paramName) {return getParam().get(paramName);}
        public Hashtable getParam() {
                if(param==null){ param = new Hashtable();  }
                return param;}}
```

Figure 4.21: Représentation du contenu d'une fenêtre.

Pour transférer la session en cours, l'usager utilise alors la fenêtre de transfert de session. (Figure 4.22 et 4.23). L'application présente à l'usager les choix qui sont disponibles. L'usager est mis au courant des endroits où les autres clients de l'agenda sont situés, de l'usager qui utilise déjà le service s'il est non disponible et d'une description sommaire de l'environnement d'utilisation du service.

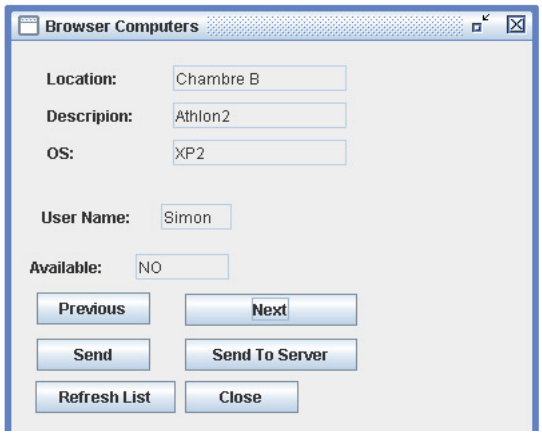

Figure 4.22: Une application agenda est occupée.

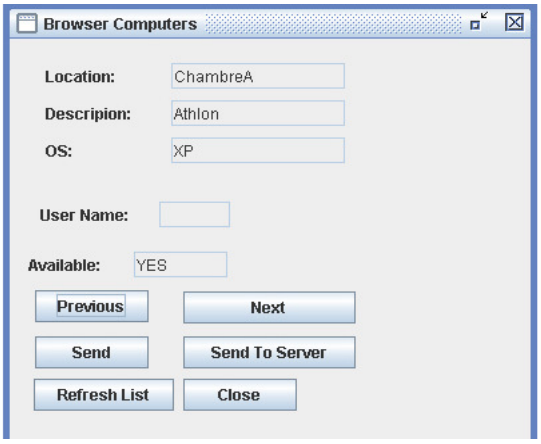

Figure 4.23 : Une application agenda est libre.

Si l'usager choisit d'envoyer la session directement sur un autre client qui est disponible, la session est transférée. L'usager peut alors se diriger vers le nouveau client de l'agenda. Le nouveau client de l'agenda présentera alors à l'usager une fenêtre pour débuter une nouvelle session. L'usager ne pourra pas changer le nom d'usager car celui-ci y sera déjà inscrit. Il suffit alors à l'usager d'y inscrire son mot de passe. Après la vérification des informations, le client de l'agenda va récréer la session en essayant de reconstituer les fenêtres, leur contenu et leur disposition. Si l'usager décide de transférer sa session sur le serveur de l'agenda, il lui suffira de débuter une nouvelle session sur n'importe quel client de l'agenda et d'effectuer la commande de récupération de la session venant d'être sauvegardée sur le serveur.

4.5 Particularités de Jini et exemples du code source des services

Pour exécuter une application java, qui est un service dans une fédération Jini, certaines particularités doivent être respectées. Une des particularités qui doit être respectée est qu'il faut spécifier en paramètre l'adresse URL pour le téléchargement du code mobile lors du lancement de l'application. Il faut donc que cette adresse soit connue avant de lancer le service. Dans le cas des services qui font parties de l'architecture Domus, cette adresse n'est pas connue à l'avance car elle est fournie par le service Domus. La figure 4.24 illustre la différence entre la ligne de commande qui lance le service agenda qui n'utilise pas le service Domus (1) et la ligne de commande qui lance l'exécution du service agenda qui utilise le service Domus pour la distribution de son code mobile (2).

```
(1) java -jar -Djava.rmi.server.codebase=http://192.168.1.50/
    -Djava.security.policy=policy.all AgendaServer.jar
(2) java –jar  -Djava.security.policy=policy.all AgendaServer.jar
```

Figure 4.24 : Ligne de commande pour lancer le service agenda.

Pour lancer un service qui ne sait pas à l'avance où sera situé son code mobile, quatre étapes sont nécessaires dans son initialisation. La figure 4.25 montre la classe MyClassLoader.java qui a été développée pour le projet. Cette classe est nécessaire puisque la propriété « *java.rmi.server.codebase* » ne peut pas être modifiée une fois que l'application est lancée et que, dans la deuxième situation du service agenda (figure 4.24), cette propriété n'a pas de valeur. On doit donc substituer la classe qui est utilisée par défaut par la machine virtuelle java par cette nouvelle classe. La figure 4.26 montre des extraits du code source de la réalisation des cinq étapes nécessaires à l'initialisation du service.

La première étape (1) est le remplacement de la classe *RMIClassLoaderSpi* par la classe de la figure 4.25. La deuxième étape est l'envoi du code mobile au service Domus qui a été découvert. La troisième étape est l'extraction de l'adresse où sera rendu disponible le code

mobile du service agenda par le service Domus. La quatrième étape est la modification de la propriété « *java.rmi.server.codebase* » par la nouvelle valeur trouvée à la troisième étape. C'est la modification de cette propriété qui ne fonctionne pas si l'on ne substitue pas la classe qui est utilisée par défaut par la machine virtuelle Java. La cinquième et dernière étape (5) est l'enregistrement du service agenda auprès de Reggie.

```
// MyClassLoader.java
public class MyClassLoader extends PreferredClassProvider
 {
     public static String CODEBASE = "";
     public MyClassLoader() { super(); }
     public MyClassLoader(boolean arg0) { super(arg0); }
     protected String getClassAnnotation(ClassLoader c)
     {
             return CODEBASE;
     }
}
```

Figure 4.25: La classe «MyClassLoader».

```
//DomusServiceProvider.java
[...]
//modification du Class Loader de RMI
System.setProperty( "java.rmi.server.RMIClassLoaderSpi",                    (1)
                    "domus.service.share.utils.MyClassLoader");
[...]
LookupDiscoveryManager ldm = getLdm();
[...]
ServiceDiscoveryManager sdm = new ServiceDiscoveryManager(ldm, null);
[...]
sdm.createLookupCache(template,null,new DomusServerListener());
[...]
DomusInfoService domus;
[...]
//On a trouvé le service Domus
//Envoie du fichier jar qui contient le code mobile du service agenda au service Domus
domus.putJar(getUserName(), getPassword(), getJar());                       (2)
String url = domus.getClassServerURL();                                     (3)
MyClassLoader.CODEBASE = url + "/" + getJar().getName();
//Cette commande ne fonctionne pas avec le ClassLoader par defaut.
System.setProperty("java.rmi.server.codebase", MyClassLoader.CODEBASE);     (4)
[...]
LookupDiscoveryManager ldm = getLdm();
ldm.addGroups(getDomusServiceProviderConfig().getGroupsToJoin());
[...]
Remote service = getProxy();  //Service à enregitrer
ServiceType serviceDesc = getServiceDesc();
Entry[] attrSets;
attrSets = new Entry[1];
attrSets[0] = serviceDesc;
[...]
//Enregistrement du service Agenda
setJoinManager(new JoinManager(service, attrSets, getServiceID(), ldm, null));  (5)
```

Figure 4.26 : Extrait du code source de l'enregistrement du service agenda.

CHAPITRE 5

MISE EN SCÈNE

Ce chapitre montre comment mettre en oeuvre les composantes développées pour réaliser un système informatique diffus. Chaque scénario couvre un aspect précis de l'informatique diffuse. La mise en commun des scénarios montre que la réalisation des systèmes informatiques diffus est à notre portée. Certains aspects ont été laissés de côté (par exemple la sécurité), non pas parce que irréalisables mais plutôt parce qu'ils ne sont pas dans le cadre de cette recherche. En outre, l'aspect intégrant la technologie Bluetooth a été mis de côté à cause de la non-disponibilité d'outils de développement en Java pour Bluetooth au moment de la réalisation. Nous commençons par présenter un scénario qui porte sur la localisation et l'identification de l'usager et sur l'adaptation du système en conséquence (§5.1). Ensuite, nous présentons un scénario de migration de sessions de travail d'un appareil vers un autre (§5.2). Nous terminons par un scénario où de l'information doit être transmise à un usager dans un environnement diffus. Dans ce scénario, plusieurs systèmes collaborent de manière transparente à l'usager. (§5.3). Tous les scénarios présentés ont été implémentés et testés.

5.1 Scénario 1 : Localisation de l'usager

Un des problèmes importants à résoudre dans un environnement diffus est la localisation de l'usager. Le scénario 1 exploré dans cette section présente l'adaptation du système face à un environnement changeant. L'agenda s'adaptera en fonction de la présence d'un usager afin de faciliter et de rendre plus agréables et naturelles les interactions avec le système informatique. Le système tente de localiser et d'identifier la personne. Ce scénario intègre les réseaux hétérogènes X10 et LAN. Nous commençons par décrire les actions de l'usager (§5.1.1), ensuite l'état initial du système (§5.1.2) et, pour finir, la description en détails de

chacune des interactions entre les différentes composantes matérielles et logicielles du laboratoire (§5.1.3).

5.1.1 Description des actions de l'usager dans le scénario 1

Dans le scénario, un usager entre dans la pièce principale du laboratoire. Le logiciel client de l'agenda affiche alors automatiquement le menu d'ouverture de session à l'écran, avec le nom de l'usager déjà inscrit. Il ne reste alors à l'usager qu'à inscrire son mot de passe pour débuter sa session. Afin de simplifier le problème et sans véritable perte de généralité, nous supposons, dans notre contexte, que c'est toujours le même usager qui entre dans la pièce et qu'il n'y a seulement qu'un seul usager présent dans la totalité du laboratoire. Cette restriction facilite l'écriture des algorithmes de détection de personne et la collaboration avec les détecteurs de mouvements. Avec cette hypothèse, on peut conclure que chaque activation des détecteurs de mouvement est toujours provoquée par le même usager. De cette façon, nous n'avons pas non plus à traiter l'identification des personnes. D'une part, il s'agit d'un problème complexe. D'autre part, nous n'avions pas de puce ou d'émetteur porté par l'usager pour l'identification. Si l'usager décide de quitter la pièce sans avoir utilisé l'agenda, le menu de démarrage de session est réinitialisé à l'état de départ.

5.1.2 État initial du scénario 1

Le scénario 1 implique le matériel suivant :

- les deux détecteurs de mouvements,
- la base réceptrice X10,
- le réseau électrique,
- le module LynX10-PLC,
- les ordinateurs A et B,
- le LAN.

Les programmes et services nécessaires sont démarrés sur les ordinateurs A et B (figure 5.1). Ils vont former les fédérations Jini pour offrir leurs services dans l'architecture Domus (figure 5.2).

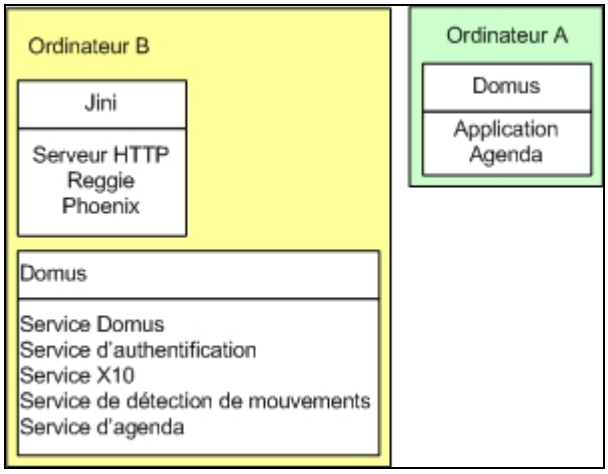

Figure 5.1: Diagramme physique des programmes pour le scénario 1.

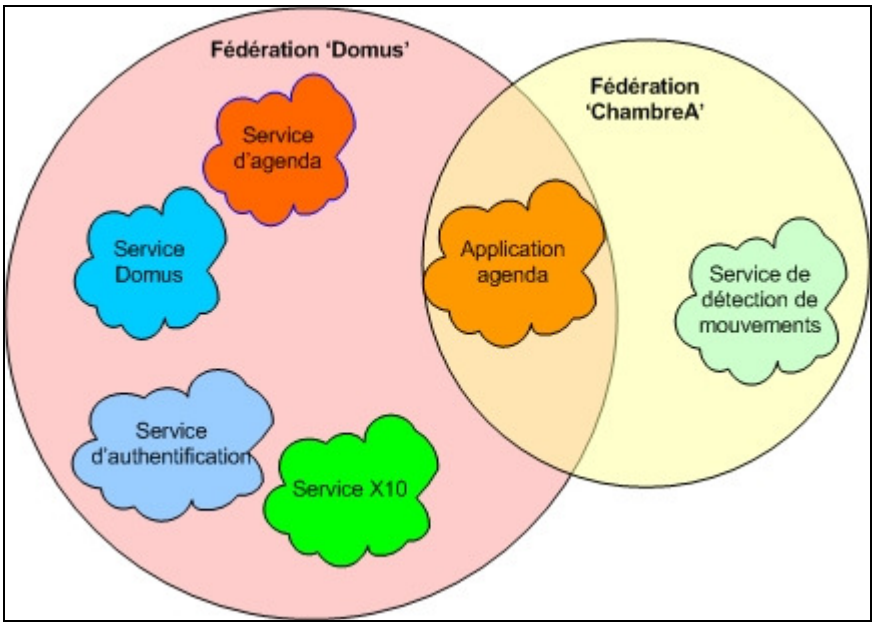

Figure 5.2: Diagramme logique des services pour le scénario 1.

5.1.3 Interactions usager – système et les flux d'informations dans le scénario 1

Une personne P entre dans la pièce (par défaut, le système croit que c'est toujours la même). Les deux détecteurs de mouvements émettent un après l'autre leurs codes de détections de présence. Le détecteur X fait face à l'extérieur de la pièce et le détecteur Y fait face à l'intérieur (figure 5.3). Le détecteur X émet donc son code radio d'identification, suivi du code d'action ON (X04B suivit de X142). Le détecteur Y émet ensuite son code radio d'identification suivi du code d'action ON (X045 suivit de X142).

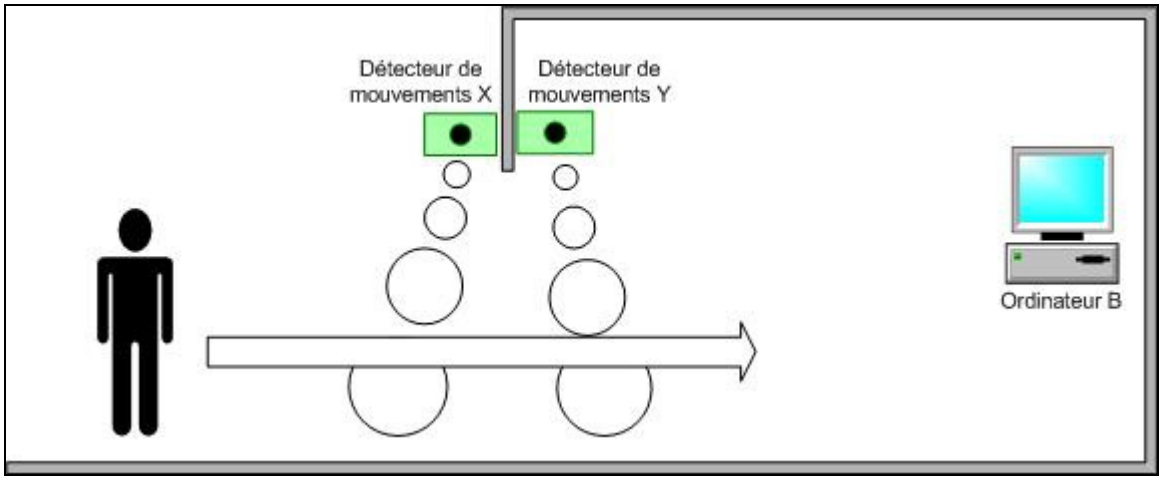

Figure 5.3: Entrée d'une personne dans la pièce.

La base réceptrice sans-fil de signaux radio capte les signaux des deux détecteurs de mouvements. Elle retransmet dans le système électrique les codes X10 des 2 détecteurs de mouvements (X04B, X142, X045, X142). Les codes X10 se propagent dans tout le système électrique de la maison pour être captés par le module LynX10-PLC. Le service X10 lit les codes provenant du module LynX10-PLC sur le port série de l'ordinateur B. Il retransmet alors les codes X10 aux services qui en ont fait la demande, en particulier le service de détecteurs de mouvements. Le service de détecteur de mouvements reçoit les codes X10 et les traite. Il détermine que quelqu'un est entré dans la pièce étant donné l'ordre d'activation des deux détecteurs de mouvements, la valeur des codes d'actions et l'intervalle de temps entre les codes reçus. Le service de détecteurs de mouvements

informe donc les clients enregistrés du changement de l'état de la pièce, en particulier l'application l'agenda. L'application de l'agenda est ainsi informé que quelqu'un vient d'entrer dans la pièce et affiche le menu de démarrage de l'application avec le nom 'Personne P' déjà inscrit. L'usager inscrit son mot de passe; une authentification s'en suit par le service d'agenda à l'aide du service d'authentification; finalement l'usager peut utiliser l'agenda (figure 5.4).

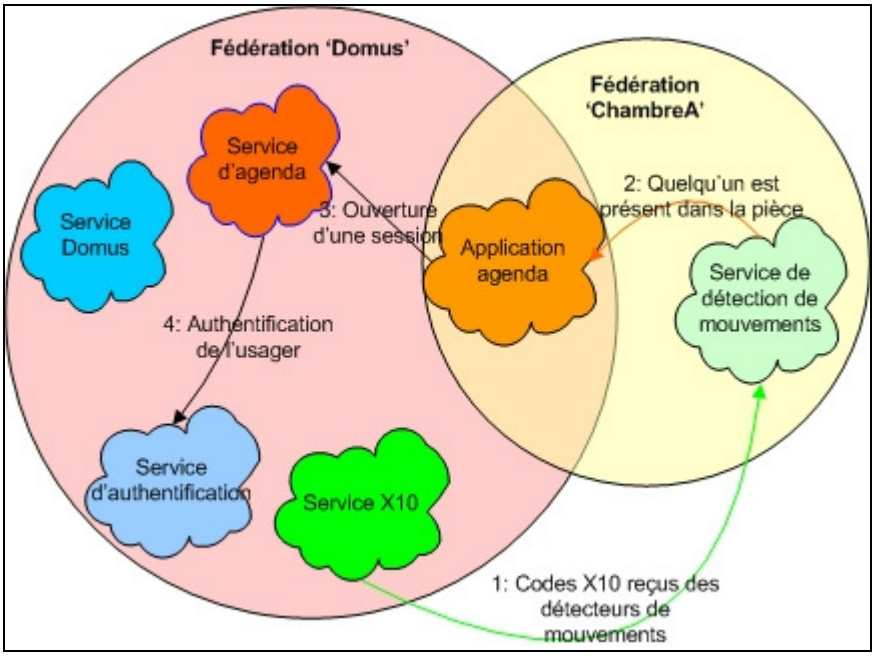

Figure 5.4: Interactions lors de l'entrée d'une personne dans la pièce.

Si la personne décide de quitter la pièce sans utiliser l'ordinateur, les détecteurs de mouvements sont activés dans l'ordre inverse. Le détecteur Y est activé et envoie son code d'identification suivi du code d'action ON (X045, X142). Le détecteur X est activé et envoie son code d'identification suivi du code d'action ON (X04B, X142). La base réceptrice retransmet dans le système électrique les codes reçus (X045, X142, X04B, X142). L'interface LynX10-PLC capte les codes X10 et les retransmet par le port série au service X10. Il reçoit les codes de l'interface LynX10-PLC et les retransmet aux clients enregistrés à ces codes (en particulier le service de détection de mouvements). Le service

de détection de mouvements reçoit les codes et détermine que quelqu'un a quitté la pièce en fonction de l'ordre d'activation, des valeurs des codes et l'intervalle de temps. Il informe ensuite les clients enregistrés que la pièce est maintenant vide, (en particulier l'application de l'agenda). L'application de l'agenda reçoit l'information que la pièce est vide et réinitialise son écran de démarrage.

5.2 Scénario 2 : La migration d'une session en cours vers un autre appareil mieux adapté

Le scénario 2 présente une manière fluide pour l'usager de changer l'appareil avec lequel il travaille. Ce changement d'appareil peut être souhaitable pour plusieurs raisons. L'usager peut vouloir travailler avec l'application dans un lieu autre que celui dans lequel il est situé, par exemple, dans une autre pièce. Il peut vouloir travailler avec un appareil possédant de meilleures caractéristiques d'affichage, par exemple, un écran de plus grande dimension. Il peut vouloir travailler sur un ordinateur plus puissant ou bien sur un ordinateur qui lui procure de la mobilité, par exemple, un PDA. Ce scénario permet à l'usager de transférer sa session de travail de manière fluide d'un appareil vers un autre, tout en reprenant là où il était rendu dans son travail avec l'application en cours. Le système tente de rendre le plus transparent possible le passage d'un appareil à un autre. Dans ce scénario, le Rubicon sémantique entre en jeu, le choix de l'appareil revient à l'usager. Nous commençons par décrire les actions de l'usager (§5.2.1), ensuite l'état initial du système (§5.2.2) et, pour finir, la description en détails de chacune des interactions entre les différentes composantes matérielles et logicielles du laboratoire (§5.2.3).

5.2.1 Description des actions de l'usager dans le scénario 2

L'usager utilise l'agenda pour y inscrire ses rendez-vous de la semaine. Il utilise l'ordinateur dans sa chambre (Chambre A). Il décide de continuer de travailler dans une autre pièce (bureau). Il choisit alors, dans l'application de l'agenda, l'ordinateur sur lequel

il désire continuer à travailler. L'agenda lui montre la liste des ordinateurs disponibles pour transférer sa session tout en indiquant si chaque ordinateur est disponible actuellement (figure 4.22). Deux cas de figure sont alors possibles :

- le transfert de session est effectué immédiatement (situation A) (§5.2.3.1);
- la session est mise en suspens pour être reprise ultérieurement (situation B) (§5.2.3.2).

Dans la première variante, l'usager choisit l'ordinateur dans le bureau et transfère la session. L'application en cours se ferme et affiche le dialogue de démarrage. L'usager se déplace dans le bureau et voit un dialogue de démarrage avec son nom d'usager déjà inscrit. Il inscrit son mot de passe dans le dialogue et la session est restaurée. L'usager peut continuer à travailler là où il était rendu dans l'application. Dans la seconde variante, par exemple advenant le cas où l'ordinateur ne serait pas disponible dans le bureau, l'usager a l'option de transférer temporairement sa session sur le serveur du service d'agenda (situation 'B'). De cette manière, l'usager pourra récupérer sa session ultérieurement lorsque l'ordinateur qui l'intéresse se libérera.

5.2.2 État initial dans le scénario 2

Le scénario 2 implique le matériel suivant :

- les ordinateurs A et B,
- le LAN.

L'état initial des deux variantes reste le même. Les programmes et les services sont démarrés sur les ordinateurs A et B (figure 5.5). Ils vont former les fédérations Jini pour offrir leurs services dans l'architecture Domus (figure 5.6).

5.2.3 Les interactions et le flux d'information dans le scénario 2

Le début des interactions est identique pour les deux variantes du scénario 2. L'usager commence par ouvrir une session sur l'agenda situé dans la chambre 'A'. Le service

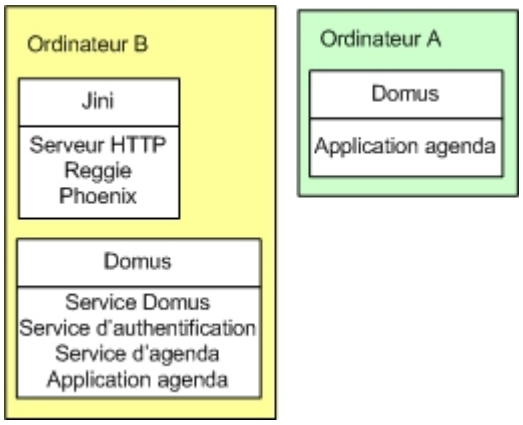

Figure 5.5: Diagramme physique des programmes pour le scénario 2.

d'agenda authentifie l'usager à l'aide du service d'authentification et la session débute. L'usager travaille dans l'agenda et ajoute des rendez-vous. À un moment donné, l'usager décide qu'il doit changer d'ordinateur. Il ouvre le menu de transfert de session et consulte la liste des ordinateurs disponibles. À cette étape, l'agenda recherche toutes les applications agenda inscrites dans la fédération Domus. Pour être inscrite, une application agenda doit déjà être en cours d'exécution sur un autre ordinateur. Cette liste est maintenue à jour grâce au réseautage spontanée. Ainsi, l'utilisateur peut consulter la liste des ordinateurs potentiellement disponibles pour y transférer la session en cours. Certains postes de travail peuvent déjà être utilisés par d'autres usagers, il sera donc impossible d'y transférer immédiatement la session en cours.

5.2.3.1 Variante A : transfert immédiat d'une session

L'usager choisit l'ordinateur dans le bureau pour transférer sa session car celui-ci est disponible. L'agenda transfère alors tous les paramètres de session nécessaires à la reconstruction sur le poste cible (figure 5.7). L'agenda met fin à la session sur le poste courant. L'usager se déplace devant le nouveau poste et trouve l'écran de démarrage avec son nom déjà inscrit. L'usager inscrit alors son mot de passe et la session est reconstruite. Le serveur agenda authentifie de nouveau l'usager à l'aide du service d'authentification (figure 5.8).

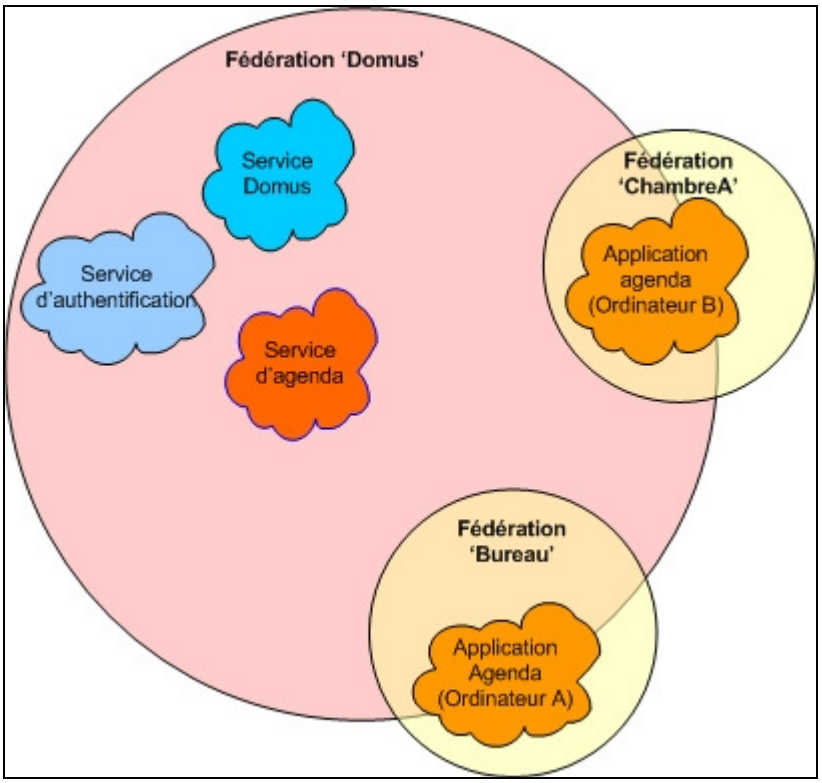

Figure 5.6: Diagramme logique des services pour le scénario 2.

5.2.3.2 Variante B : sauvegarde d'une session pour reprise ultérieure

Le poste que l'usager désire n'est pas disponible. L'usager choisit alors de transférer la session sur le serveur pour la récupérer plus tard (figure 5.9). L'agenda transfère tous les paramètres nécessaires à la reconstruction de la session sur le serveur et termine ensuite la session en cours. Lorsque le moment est propice, l'usager se dirige vers l'ordinateur de son choix pour continuer sa session. Il débute alors sa session en inscrivant son nom et mot de passe dans l'agenda choisi. Le service d'agenda authentifie de nouveau l'usager à l'aide du serveur de sécurité.

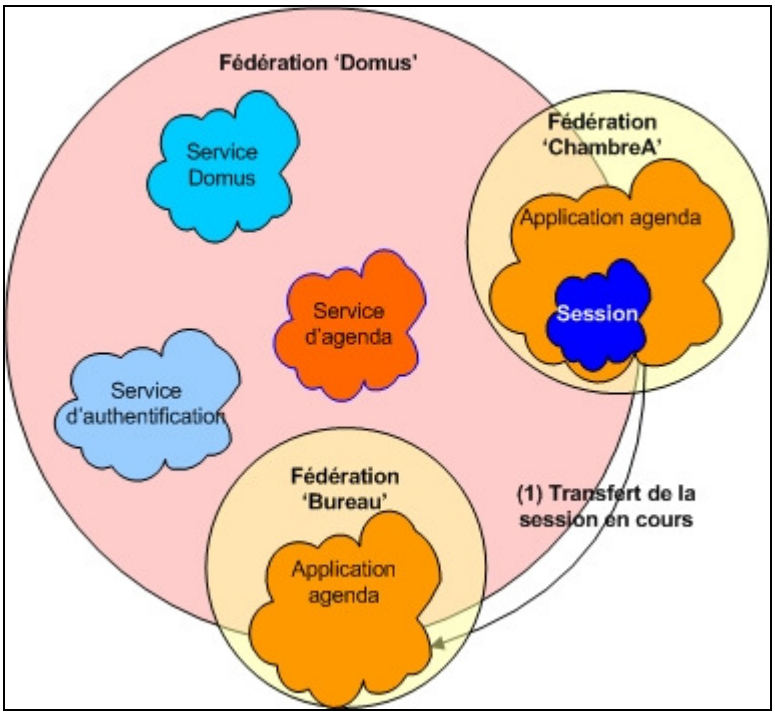

Figure 5.7: Première étape du transfert de session.

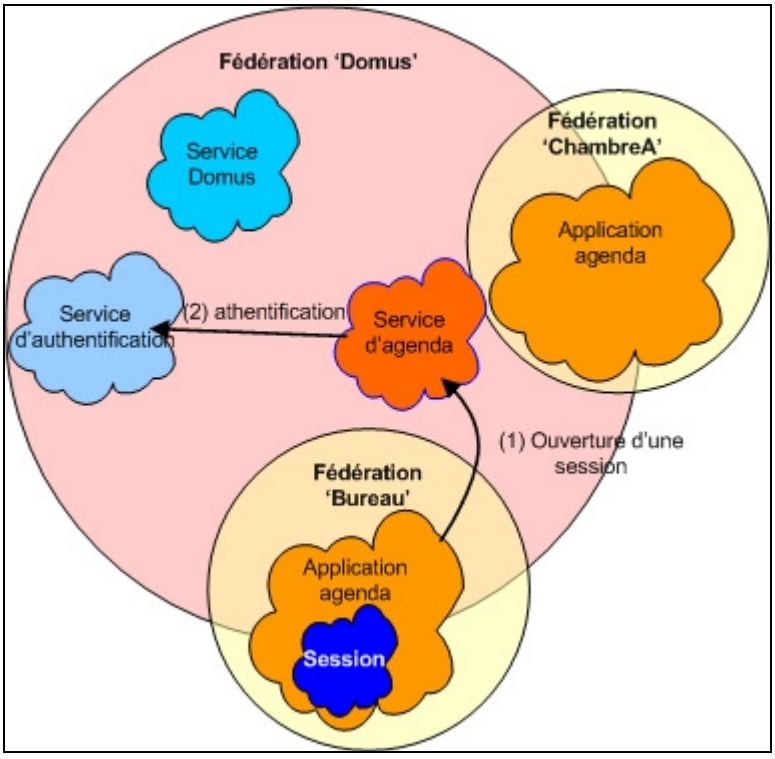

Figure 5.8: Deuxième étape du transfert de session.

Une fois la session initiée, l'usager choisit à l'aide des commandes, de reprendre la session précédente. L'agenda récupère les paramètres de session qui sont sur le serveur et reprend la session. L'usager peut ensuite continuer à travailler là où il était rendu (figure 5.10).

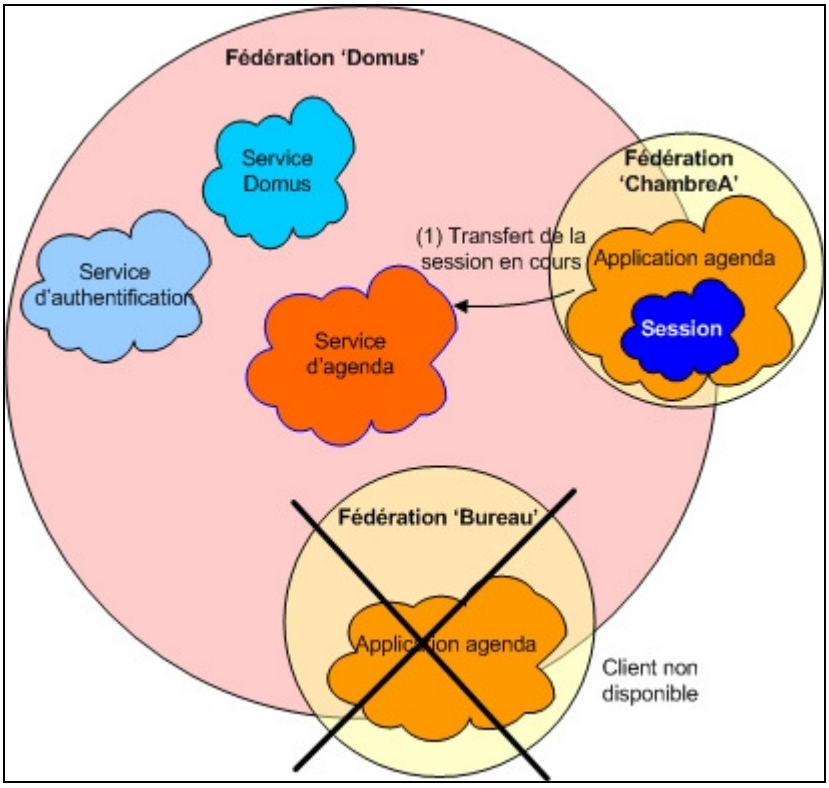

Figure 5.9: Transfert de la session sur le serveur.

5.3 Scénario 3 : Le rappel d'un rendez-vous

Dans le troisième scénario, le service d'agenda essaie d'avertir un usager qu'un rendez-vous approche. La procédure d'avertissement comporte deux étapes. La première étape est un avertissement qui a lieu un long moment avant le rendez-vous (quelques jours). La deuxième étape est un avertissement qui a lieu un court moment avant le rendez-vous (quelques heures). Ces deux étapes sont réalisées de la même manière. Pour cela, le serveur agenda se sert du service de messagerie qui possède de meilleurs moyens pour

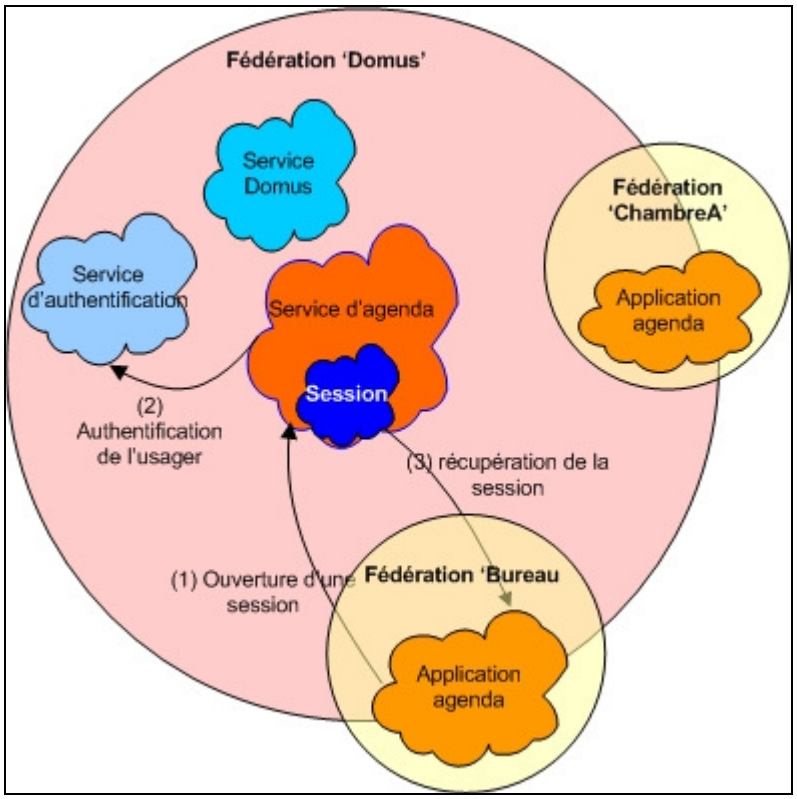

Figure 5.10: Récupération de la session sauvegardée sur le serveur.

contacter un usager. Lorsque l'usager reçoit un message d'avertissement, il entre son mot de passe et peut lire le message. En faisant cela, il confirme au système la réception du message et le service d'agenda sait que l'usager a été averti de l'approche de son ou ses rendez-vous. Lorsque le service d'agenda reçoit la confirmation de réception du premier avertissement, il attendra le temps venu pour transmettre une demande de message au service de messagerie pour le deuxième avertissement.

Le troisième scénario s'appuie encore une fois sur le réseautage spontané et les services de découverte. Il poursuit aussi l'objectif d'assurer un état propre et cohérent de l'ensemble de l'environnement diffus. Cet aspect est très important à préserver dans un environnement où les applications vont et viennent dans l'ensemble du système et où les applications peuvent se déplacer. Une application qui se déplace pourrait être le service d'affichage de message qui s'exécute sur un iPAQ avec une connexion au réseau grâce à un premier

routeur sans-fil. Par la suite, l'usager se déplace avec son iPAQ hors de la portée du premier routeur sans-fil pour être reconnecté au réseau par un deuxième routeur sans-fil avec une nouvelle adresse IP. Dans notre contexte, le terme « état propre » signifie qu'il n'y a pas de messages inutiles ou périmés qui traînent dans l'environnement. Ainsi l'agenda reste en contrôle des messages qui sont affichés. Nous commençons par décrire les actions de l'usager (§5.3.1), ensuite l'état initial du système (§5.3.2) et, pour finir, la description en détails de chacune des interactions entre les différentes composantes matérielles et logicielles du laboratoire (§5.3.3).

5.3.1 Description des actions de l'usager dans le scénario 3

L'usager commence par ouvrir une session dans le client agenda. Il entre son nom et son mot de passe. Ensuite, il ajoute ses rendez-vous en spécifiant la date et l'heure de chaque rendez-vous ainsi qu'une brève description (figure 5.11). Il peut par la suite quitter l'application. Lorsque le temps est venu, le système tente de rappeler à l'usager qu'un rendez-vous approche. L'usager pourra voir sur un des écrans de sa maison (dans le laboratoire) qu'un message lui est destiné et qu'il peut le lire s'il le désire. Lorsque son attention sera captée par un des récepteurs de messages, l'usager entre son mot de passe pour pouvoir lire le message (figure 5.12). L'usager lit son message, ferme la fenêtre et continue ses activités (figure 5.13).

5.3.2 État initial dans le scénario 3

Le scénario utilise le matériel suivant :

- le LAN,
- les ordinateurs A et B.

Plusieurs services s'exécutent sur chacun des ordinateurs (figure 5.14). Ils vont former les deux fédérations de base (figure 5.15).

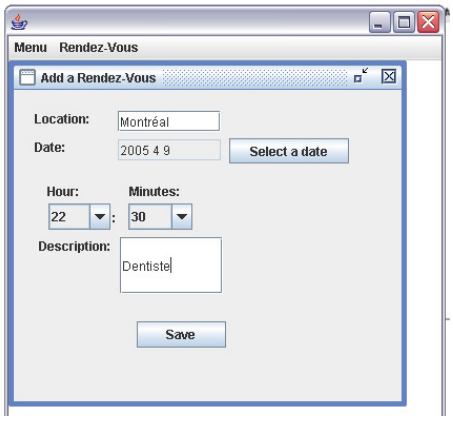

Figure 5.11: Ajout d'un rendez-vous

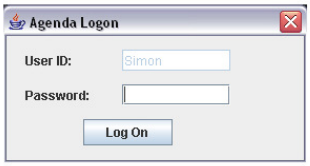

Figure 5.12: Écran de démarrage du récepteur de message

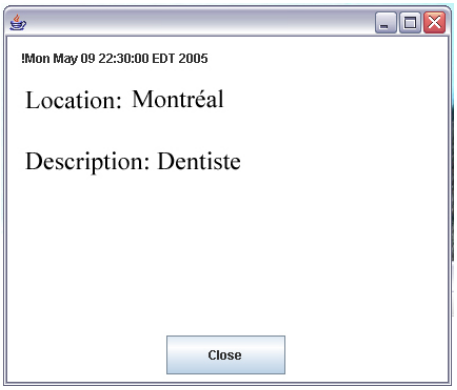

Figure 5.13: Fenêtre du rappel du rendez-vous

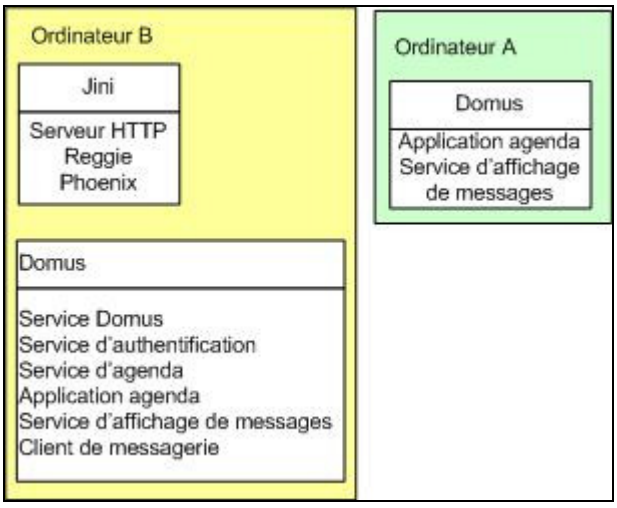

Figure 5.14: Diagramme physique des programmes du scénario 3.

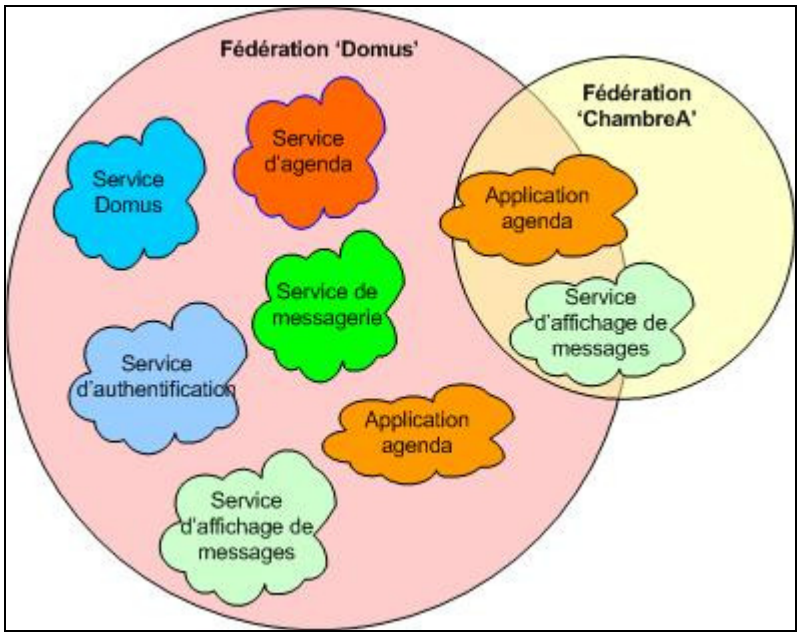

Figure 5.15: Diagramme logique des services du scénario 3.

5.3.3 Les interactions et le flux d'information dans le scénario 3

L'usager démarre une session sur une application agenda. Il entre son nom et mot de passe. Le serveur agenda vérifie les informations en utilisant le service d'authentification. L'usager ajoute des rendez-vous (figure 5.16) et termine la session. À intervalles prédéterminés, le service d'agenda vérifie la base de données pour trouver des rendez-vous qui nécessitent un avertissement. Une fois cette liste créée, le serveur agenda recherche un serveur de messagerie. Une fois un serveur trouvé, le serveur agenda élabore un message contenant les détails du rendez-vous. Il envoie ensuite ce message au service de messagerie avec le nom du destinataire (figure 5.16). Après cette étape, le service d'agenda attend une confirmation de réception du message par le destinataire de la part du service de messagerie. Dans ce scénario, on ne tente pas de prédire la position de l'usager. Sa position est déduite d'après le service qui effectue la confirmation de réception du message par l'usager. Le service d'agenda doit aussi conserver une liste de toutes les demandes de livraisons de messages qu'il a faites au service de messagerie. Avec cette liste, il peut gérer l'état des rendez-vous et l'état des demandes de livraisons de messages. Le service

d'agenda doit faire une demande de retrait de message s'il considère qu'il n'est plus nécessaire de contacter un usager ou si le rendez-vous est périmé afin de conserver le système diffus dans un état propre. Lorsque le service de messagerie reçoit une demande d'envoi de message, il recherche tous les services de réception de messages. Lorsque ceux-ci sont trouvés, il leur envoie le message reçu avec le destinataire (figure 5.16). Le service de messagerie doit aussi gérer les récepteurs de messages qui se joindraient à la fédération après la première recherche afin de leur envoyer les messages qui auraient été manqués. Ce dernier comportement montre la capacité du système à évoluer dans un environnement où les appareils peuvent se joindre ou quitter dynamiquement l'environnement.

Chaque récepteur affiche alors le message de manière indépendante. Le récepteur de message affiche en particulier un dialogue avec le nom du destinataire et attend que ce dernier inscrive son mot de passe. Lorsque l'utilisateur confirme, c'est au récepteur de message d'authentifier l'usager avec le service d'authentification. L'usager peut ensuite lire son message et après la lecture, fermer la fenêtre dans laquelle le message est affiché. Une fois le message lu, le service d'affichage de messages contacte le service de messagerie qui lui a distribué le message pour lui confirmer l'acheminement du message avec succès (figure 5.17). Lorsque le service de messagerie reçoit une confirmation, il doit retirer ce message qui est possiblement encore affiché sur les autres services d'affichage de messages. Le service de messagerie contacte alors les services d'affichage de messages et leur communique l'identificateur du message à retirer. Le service de messagerie gère aussi les récepteurs de message déconnectés qui se joindraient à nouveau à la fédération et qui auraient manqué la demande de retrait du message. Lorsque ceux-ci se reconnectent au système, le service de messagerie leur communiquera tous les messages qui doivent être retirés. Le service de messagerie confirme finalement au service d'agenda que le message fut délivré avec succès en lui donnant aussi des informations sur la réception du message (figure 5.18). Ces informations sont l'heure de la confirmation, l'identité du service qui a délivré le message, etc. Le serveur agenda met ensuite à jour ses rendez-vous confirmés et est prêt pour les prochaines opérations.

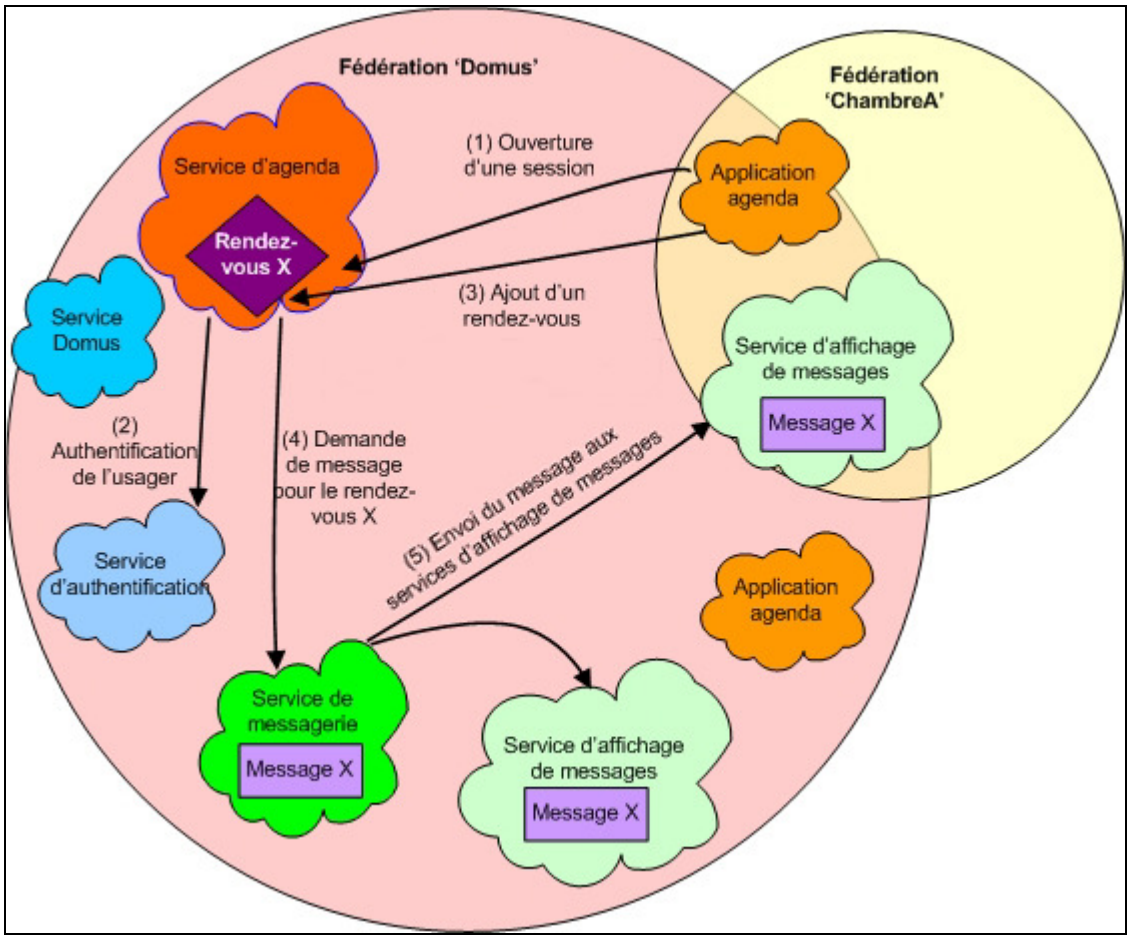

Figure 5.16: Distribution des messages.

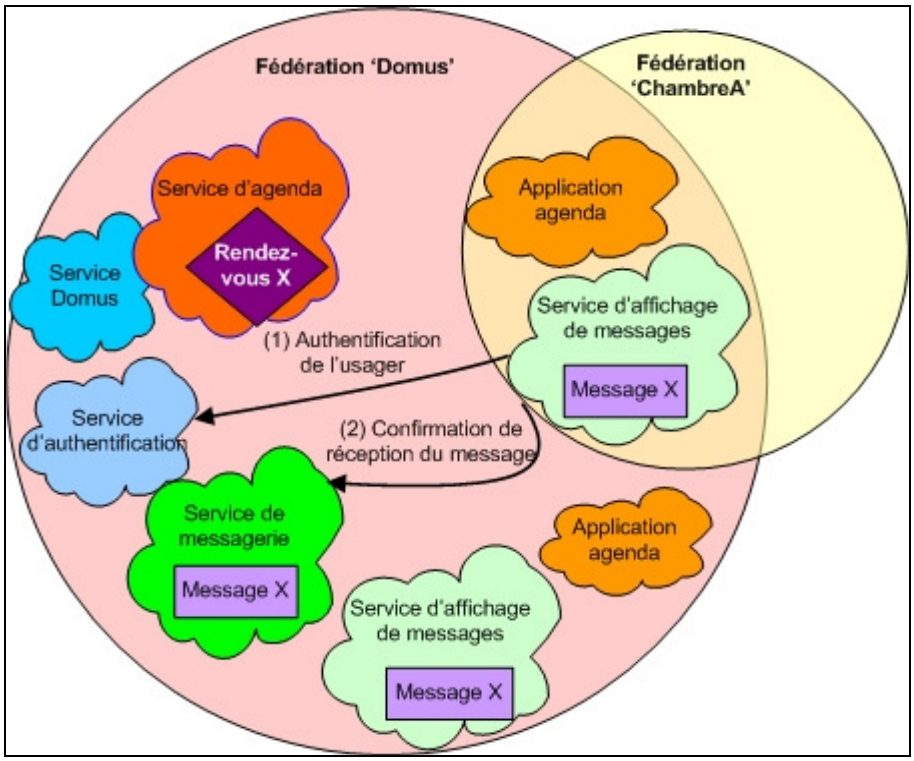

Figure 5.17: Réception par l'usager.

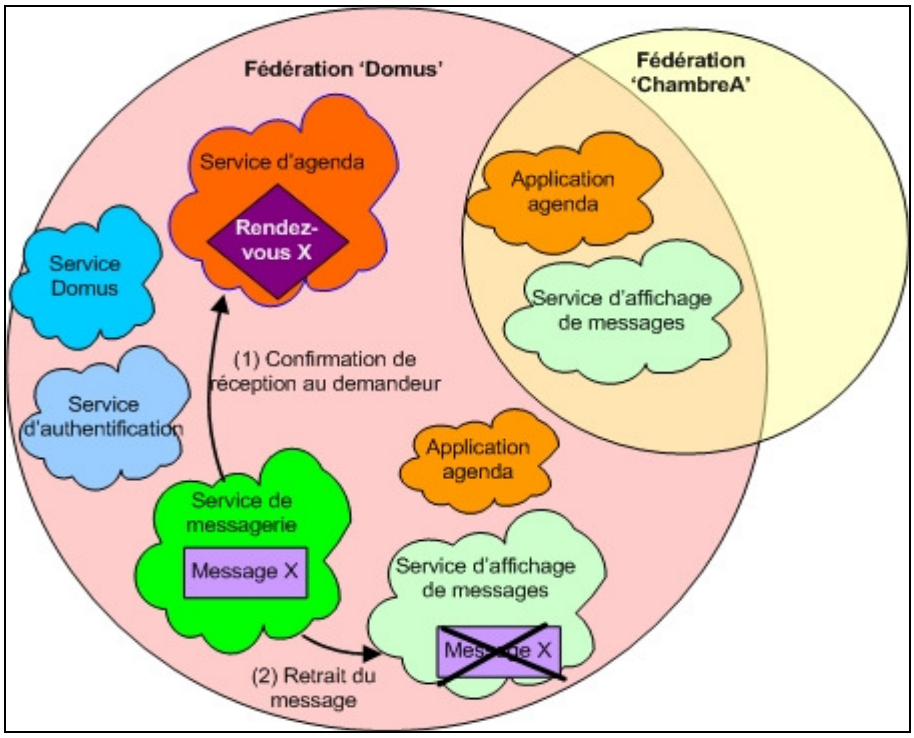

Figure 5.18: Confirmation de réception.

CHAPITRE 6

COMPARAISONS AVEC DES SYSTÈMES EXISTANTS

Ce chapitre compare les composantes développées pour réaliser le système informatique diffus avec d'autres systèmes informatiques qui ont des caractéristiques et des motivations semblables. Nous commençons par comparer le système Domus avec la plateforme informatique diffuse ICrafter (§6.1). Ensuite, nous le comparons avec l'outil de localisation de personnes People Finder (§6.2). Nous poursuivons par la comparaison du système CARE™ [11] développé et mis en opération chez Elite Care à Milwaukie (Oregon) avec Domus (§6.3). Nous terminons par la comparaison du système VNC (§6.4) avec Domus. La comparaison entre les systèmes est faite en fonction des caractéristiques des systèmes diffus, c'est à dire sur l'hétérogénéité, l'adaptation, l'intégration physique, la collaboration, la flexibilité, la robustesse et la répartition.

6.1 ICrafter

ICrafter [11] est une plateforme informatique diffuse qui permet à l'usager d'interagir avec des appareils et des services situés dans un lieu de travail interactif. Un lieu de travail interactif est un endroit physique où cohabitent des appareils électroniques (imprimantes, numériseurs électriques), des ordinateurs et des appareils possédant des caractéristiques d'entrées/sorties (microphones, écran, haut-parleurs) avec des gens qui se rencontrent pour effectuer un travail de collaboration. Un bon exemple d'un lieu de travail interactif est une salle de conférence. Les appareils peuvent être des projecteurs et des microphones. Les services peuvent être tout ce qui est utile à l'usager comme des appareils (une lampe, un projecteur ou une imprimante) ou des applications (fureteur Web, Microsoft Power Point). Le système ICrafter génère automatiquement les interfaces usager pour les appareils et peut

[11] Care™ : Creating an Autonomy-Risk Equilibrium

aussi générer automatiquement des interfaces composées de plusieurs appareils. Ces interfaces composées sont généralement des agrégations producteur/consommateur comme une combinaison caméra (producteur)/imprimante (consommateur). Tout comme l'architecture Domus, ICrafter est conçu pour évoluer dans un environnement hétérogène.

Une des principales caractéristiques de ICrafter est qu'il génère automatiquement des interfaces graphiques pour les appareils, ce que la plateforme Domus ne fait pas. ICrafter utilise un service qui est très centralisé pour la génération d'interfaces graphiques. Tous les appareils de ICrafter utilisent ce générateur d'interfaces graphiques pour être représentés lorsqu'ils sont utilisés. Cette méthode de génération d'interface graphique est donc très centralisée, mais le générateur peut être répliqué pour augmenter la flexibilité et la robustesse. La plateforme Domus n'a pas pour but de présenter des interfaces graphiques pour contrôler les appareils par un usager. La plateforme Domus va plutôt contrôler un appareil sans l'intervention d'un usager dans le but de réaliser une tâche, donc aucune interface graphique n'est nécessaire pour l'usager. La plateforme ICrafter est plutôt passive et attend les commandes d'un usager tandis que la plateforme Domus va plutôt être active pour adapter l'environnement face aux utilisateurs afin d'être complètement intégrée et invisible. Pour qu'un système soit le plus diffus possible, doit-il présenter une interface informatique pour ouvrir une lumière ou est-il plus naturel d'utiliser l'interrupteur ? Le système sera encore plus diffus si aucune intervention de l'usager n'est nécessaire et que le système accomplisse certaines tâches, par exemple la lumière qui se ferme lorsque l'on quitte une pièce qui devient vide. Par contre, il faut garder en tête qu'il est parfois trompeur de vouloir deviner ce que veut l'usager.

La découverte des services ressemble à la méthode utilisée par Jini. Pour annoncer un service, ICrafter utilise un 'beacon' qui indique l'emplacement exact d'un service. Donc, si ce service a une défaillance et se rétablit rapidement, le *beacon* indiquera encore la bonne adresse du service. Par contre dans l'architecture Jini, le système utilise un proxy. Ainsi, si un service Jini subit une défaillance et se réenregistre, il y aura un nouveau proxy pour annoncer le service. L'ancien proxy est seulement retiré à la fin de son bail. Cela fera

donc deux proxys d'enregistrer dans Reggie pour le même service, un proxy étant obsolète et l'autre non. Il faudra attendre que le système de baux fasse le ménage et enlève le proxy associé au service défaillant. ICrafter est plus robuste en général pour cette raison par rapport à Jini. Toutefois dans notre implémentation actuelle, nous vérifions si le service reçu est défaillant ou non avant de l'utiliser, donc la robustesse nous apparaît à peu près semblable à celle offerte par ICrafter. Cette robustesse est implémentée au niveau de Domus et non au niveau de Jini.

Finalement les deux plateformes utilisent le langage Java, toutefois ICrafter utilise aussi d'autres plateformes pour l'exécution des interfaces graphiques (Tcl et Python). ICrafter utilise aussi plusieurs langages pour la description des interfaces usager des appareils (HTML, VoiceXML, MoDAL et SUIML). Domus n'utilise pas d'autres langages pour décrire des services et utilise seulement la plateforme Java et J2ME.

6.2 PeopleFinder

PeopleFinder [6] est un outil qui utilise des agents pour déterminer indirectement la position des usagers dans un lieu de travail. Les bases de PeopleFinder reposent sur la plateforme CALVIN qui elle-même est une extension de l'architecture TouringMachine. C'est un système composé d'agents autonomes. Chacun est spécialisé pour réaliser une ou plusieurs fonctions. Le système Domus est aussi composé de services qui sont spécialisés dans une ou plusieurs fonctions comme le service de messagerie et le service d'affichages de messages.

Les agents autonomes sont regroupés dans trois catégories. La première catégorie regroupe les agents qui s'occupent des tâches reliés aux usagers du système comme l'interprétation de commandes vocales, l'envoi de courriel ou la signalisation sur un téléphone. La deuxième catégorie regroupe les agents qui s'occupent des tâches au niveau de l'application comme mettre à jour une interface graphique ou appliquer des algorithmes de

décision sur la manière de contacter un usager selon ses préférences. La troisième catégorie regroupe les agents qui s'occupent des tâches au niveau du système d'exploitation comme la traduction entre différents encodages de la voix. Domus possèdent aussi différents niveaux de services mais dans certains cas la délimitation n'est pas aussi claire que dans PeopleFinder. L'architecture Domus possède des services reliés aux actions des usagers (première catégorie) comme l'application agenda. Cette application est utilisée en majeure partie par un usager qui ajoute, supprime et consulte ses rendez-vous. Domus possède des services au niveau applicatif (deuxième catégorie), comme le service d'agenda. Ce service appartient à la deuxième catégorie puisque la majorité des interactions avec ce service se font à partir d'autres services ou applications (service de messagerie, application de l'agenda). Domus possède aussi des service de la troisième catégorie, comme le service X10 qui s'occupe de la communication avec le port série et l'interface X10 et qui intègre le service à l'architecture Domus. Par contre le service d'affichage de messages appartient à la fois au premier et au deuxième niveau car les interactions se font autant par un usager que par un autre service. La partie de l'interaction avec l'usager (interface graphique, saisie du mot de passe et l'affichage du message) appartient au premier niveau tandis que l'aspect du service qui accepte et qui gère les demandes de messages appartient au deuxième niveau. L'outil PeopleFinder est composé d'un nombre d'agents autonomes coordonnés entre eux tandis que le système Domus est composé de services qui sont distribués.

Tout comme le système PeopleFinder, Domus peut recevoir des données provenant des usagers (service de détection de mouvements, application agenda) et possède des moyens d'informer les usagers (service de messagerie). Le système PeopleFinder possède plusieurs types de médias pour l'interaction avec l'usager comme l'usage d'interfaces graphiques, de son et de vidéo. Domus utilise principalement les interfaces graphiques et le texte mais rien n'empêche le développement de services qui utilisent ces caractéristiques sur de nouveaux appareils ou sur des appareils déjà existants. Un nouveau service avec ce genre de capacité peut même être intégré de manière transparente avec les autres services grâce aux caractéristiques de Jini.

Tout comme Domus, PeopleFinder intègre plusieurs applications hétérogènes qui s'exécutent sur des plateformes et des appareils hétérogènes.

6.3 Care™

L'architecture informatique décrite dans l'article «Using Pervasive Computing to Deliver Elder Care » [15] s'intéresse à la même population que celle visée par notre système. Le système présenté dans cet article est très intégré à l'environnement. Il est orienté vers l'utilisation de senseurs pour enregistrer certains signes vitaux et localiser les usagers. L'infrastructure informatique est plutôt du type statique contrairement à Domus qui utilise le réseautage spontané. Les applications qui sont à la disposition des usagers sont aussi standards, en particulier, elles ne coopèrent pas pour effectuer une tâche précise. L'infrastructure informatique mise en place est imposante et est composée de beaucoup de matériel spécialisé comme des senseurs et des émetteurs sans-fil que les usagers portent sur eux. Beaucoup de matériel informatique y est présent. Le filage (30 milles) et autres éléments ont été intégrés dès la construction du bâtiment. Cet aspect rend l'informatique très intégrée physiquement avec le bâtiment au point de vue esthétique et physique. L'intégration de l'informatique se fait de manière invisible pour l'usager. Malgré une architecture statique et aucune interaction spontanée, le système s'adapte quand même aux conditions changeantes lorsqu'un usager situé à l'intérieur des bâtiments et se déplace vers l'extérieur. La méthode de localisation par l'émetteur utilise le mode infrarouge à l'intérieur des bâtiments et le mode radio à l'extérieur. Tout comme Domus, Care a été développé avec l'intention d'intégrer des technologies informatiques à faible coût. Au point de vue de l'informatique mobile, Care intègre seulement des émetteurs d'identification tandis que Domus utilise la technologie des réseaux sans-fil avec le iPAQ.

6.4 Virtual network computing

Le système VNC [13] (virtual network computing) a pour but de reproduire le bureau d'un usager depuis n'importe quel station de travail située n'importe où dans le monde. C'est exactement l'inverse d'accéder à n'importe quel système situé n'importe où dans le monde depuis notre poste de travail. Ce système va récréer le bureau de l'usager dans les moindres détails et utilise le moins de ressources possibles. Tout le travail est réalisé du coté serveur. Le but de ce système se rapproche un peu du scénario de migration de session. Dans VNC, les clients ne conservent aucun paramètre, tout est calculé par le serveur central. Dans Domus, il n'y a pas de serveur central et les applications se transmettent directement les paramètres pour reproduire une session le plus fidèlement possible.

Le système VNC fonctionne sur plusieurs plateformes hétérogènes, mais il faut un serveur pour chaque combinaison de bureau/client. Il n'y a pas de découverte spontanée de services étant donné que chaque utilisateur n'accède qu'à son propre bureau. Les clients sont très simples, tout le travail est fait du côté serveur qui doit aussi tenir compte de la bande passante entre le serveur et le client pour que l'interface graphique soit le plus convivial possible.

CONCLUSION

Cette conclusion montre comment l'architecture Domus et le système diffus de rappel de rendez-vous ont réussi à intégrer beaucoup de facettes de l'informatique diffuse. Dans cette discussion, nous abordons les caractéristiques des systèmes diffus qui ont été établies au début du mémoire. Nous commençons par présenter l'intégration physique, l'inter-opération spontanée, la découverte et l'interaction, l'adaptation, l'intégration, le cadre de programmation, la robustesse et la sécurité. Nous discutons ensuite de l'architecture Domus pour terminer avec les travaux futurs à envisager.

L'intégration physique est réalisée par l'utilisation de détecteurs de mouvements et de détection d'éclairage. L'utilisation d'équipements qui fournissent des informations physiques sur l'état de l'environnement au système informatique a quand même été tenue au strict minimum. Si le besoin s'en fait sentir, il serait très facile d'ajouter les senseurs appropriés pour qu'ils facilitent l'exécution d'une tâche ou bien qu'ils facilitent la prise de décision d'un côté ou de l'autre du Rubicon sémantique. L'aspect de l'identification exacte de la personne n'a pas été réalisé. On peut détecter la présence d'une personne, mais on ne peut pas faire la différence entre une ou plusieurs personnes qui seraient présente dans le laboratoire. On ne peut pas non plus faire la différence entre une personne et un animal de compagnie avec l'équipement utilisé.

Le réseautage spontané a été utilisé à plusieurs reprises, par exemple la découverte de client de l'agenda pour la migration de session et la recherche de récepteurs de messages. Le Rubicon sémantique a été abordé quelques fois, mais il reste quand même difficile à cerner. Lorsque l'usager quitte la pièce, c'est le côté système du Rubicon sémantique qui prend la décision de fermer l'écran de démarrage. Lorsque l'usager choisit l'appareil où migrer sa session, c'est du côté usager que la décision est prise.

La découverte et l'interaction pour réaliser une tâche sont faites dans tous les scénarios. Le serveur de messagerie utilise les récepteurs de client, le serveur agenda utilise le serveur de messagerie et le serveur de sécurité, etc. La découverte et l'interaction est faite grâce à la technologie Jini et RMI.

L'adaptation se réalise lors de la migration de session vers un appareil avec moins de ressources comme lorsque la session se transfère sur un PDA. Malgré le fait qu'il a fallu faire un sous-projet pour intégrer la technologie Madison, l'adaptation a quand même été réalisée au niveau global. Étant donné que Madison fonctionnait sous la version 1.2 de Jini, le sous-projet était constitué des mêmes services que le projet régulier mais la version de Jini utilisée était la version 1.2 au lieu de la version 2.0. La raison qui limitait la technologie Madison dans son intégration est due au fait que la version Jini 2.0 n'est pas rétro-compatible avec la version Jini 1.2. Le langage utilisé dans les architectures Jini 2.0 et Jini 1.2, ainsi que dans Madison est Java 2.0. Il suffit d'adapter le logiciel Madison à la version courante de Jini pour qu'il fonctionne. Les versions de Java et de Jini qui sont utilisées dans l'architecture Domus ne sont pas non plus des facteurs de limitatifs face à la version de J2ME qui s'exécute sur le iPAQ puisque celui-ci communique avec des datagrammes sur la connexion réseau (UDP).

L'hétérogénéité des réseaux a été unifiée par l'utilisation du langage Java et de l'architecture Jini (X10, LAN, WiFi). Le réseau électrique et les composantes qui y sont rattachés interagissent avec des composantes qui utilisent le réseau LAN et sans-fil. L'hétérogénéité des systèmes d'exploitation a aussi été unifiée (Windows vs Linux). Le code nécessaire à l'exécution de l'application agenda est exactement le même code qui s'exécute dans l'environnement Windows que dans l'environnement Linux grâce au langage Java. L'hétérogénéité des appareils a aussi été unifiée (iPAQ vs PC). On a pu migrer les données d'une session du client de l'agenda qui était sur l'ordinateur A (Windows XP) sur l'ordinateur B (Linux) et sur le PDA (PocketPC 2003) et vice-versa. Les services ont communiqué par différents types de réseaux, c'est-à-dire les réseaux sans-fil, LAN et électriques (X10) pour réaliser leurs tâches. Au niveau de l'architecture

Domus, les services sont rendus homogènes au niveau logiciel malgré le fait que des appareils hétérogènes et des réseaux hétérogènes collaborent pour réaliser une tâche.

L'intégration d'équipements informatiques de différentes générations est réalisée avec l'utilisation des deux ordinateurs. L'intégration de systèmes d'exploitation différents a aussi été réalisée avec l'utilisation de Linux et Windows XP. On aurait pu intégré un ordinateur qui utilise un système d'exploitation comme Windows 95. Il faut concevoir le système en fonction d'obtenir le meilleur compromis entre les versions de Java et de Jini disponibles sur chacune des plateformes afin de pouvoir intégrer le plus grand nombre d'appareils utiles à la réalisation de l'architecture. Il faut trouver une base minimale pour fonder le système et préparer son évolution.

La robustesse est intégrée grâce à l'architecture Jini et au niveau de la programmation des services développés pour Domus. Cette robustesse est à la fois présente au niveau de Jini et au niveau des services de Domus. Au niveau de Jini, la robustesse est implémentée avec les services activables et persistants de Reggie. Cette robustesse est aussi intégrée de manière transparente puisque le système est réparti sur plusieurs ordinateurs et dans plusieurs services indépendants. Jini utilise un service de baux qui assure une certaine intégrité des services enregistrés auprès de Reggie. Au niveau applicatif, les services peuvent récupérer d'une utilisation d'un proxy défaillant (qui est le lien avec un service qui s'exécute sur un autre ordinateur) grâce aux fonctionnalités du langage (try/catch) et de la logique de la programmation des services.

La sécurité est abordée avec le service d'authentification. Certains aspects de la sécurité peuvent être considérés comme des aspects standards où il faut seulement appliquer les techniques déjà existantes (transmission sécurisé, sécurité lors de l'exécution des applications Java) mais d'autres aspects sont tout à fait nouveaux et ils ne sont pas traités de la même manière. D'autres aspects techniques restent à être explorés comme la migration d'un réseau local vers un autre. Cette migration pose un problème de configuration de l'appareil puisque les systèmes d'exploitations n'ont pas de moyens de

choisir automatiquement le bon réseau et/ou la bonne configuration lorsque plusieurs réseaux sont disponibles. Cette migration peut causer des problèmes autant au point de vue d'un PDA qu'avec un ordinateur portable. Aussi, avec la prolifération des réseaux sans-fil, un usager peut avoir une certaine difficulté à adapter son appareil à la bonne configuration réseau et il peut être confondu par la présence de plusieurs réseaux qui sont présents mais qui ne sont d'aucune utilité (comme par exemple le réseau sans-fil d'un hôtel avoisinant). La venue d'appareils sans-fil qui sont étrangers au système pose aussi un problème d'ordre technique et aussi de sécurité. L'adhésion de nouveaux services provenant de sources étrangères au système peut aussi causer des brèches de sécurité. La possibilité de boucles infinies entre les services est aussi une possibilité envisageable.

Architecture Domus

L'architecture Domus réussit à intégrer beaucoup des composantes offertes par un environnement Jini de manière efficace. Les applications de Domus utilisent Reggie pour localiser les services dont ils ont besoin. Les services utilisent aussi Reggie pour s'enregistrer et ainsi se rendre disponibles aux applications pour que celle-ci réalisent leurs tâches. Les services utilisent le renouvellement de baux pour rester enregistré aux fédérations. L'utilisation des baux assure un état cohérant à l'architecture Domus. Les applications interagissent entre elles grâce à RMI et à la «marshalisation» des données. L'architecture Domus utilise aussi la programmation par évènements distribués. Cette technique est utilisée par le service de détection de mouvements pour informer l'application agenda qu'un usager pénètre dans la pièce principale. Le déploiement des services atteint un niveau de flexibilité encore plus grand grâce au service général Domus qui permet à un service de ne pas connaître à l'avance l'emplacement de son code mobile. Cette caractéristique n'est pas une fonctionnalité déjà présente dans l'architecture Jini et est unique à l'architecture Domus. L'architecture Domus réussit à intégrer des applications et services (l'application agenda et le service d'affichage de messages qui s'exécutent sur le iPAQ) qui sont programmés dans un langage différent et incompatible avec la technologie

RMI de Java (on ne peut pas utiliser la technologie RMI entre le Java et le J2ME) de manière transparente à l'architecture Jini grâce à Madison. Finalement, grâce à Java, l'environnement Jini et l'architecture Domus, on a réussi à rendre très homogène au niveau logique plusieurs appareils et plateformes informatiques ainsi qu'à simplifier la tâche de développement et de déploiement de services dans divers environnements hétérogènes. Grâce à Jini et à Java, le développement d'applications pour les environnements diffus peut être réalisé en faisant entièrement abstraction des couches matérielles et des systèmes d'exploitations.

Les travaux futurs

Les améliorations au système se divisent en trois axes principaux. Le premier axe est l'identification des individus (§7.9.1), le second axe porte sur la localisation physique des individus et des services (§7.9.2) et, pour terminer, le dernier axe porte sur les alternatives à Jini (§7.9.3).

7.9.1 L'identification des individus

Les travaux futurs devraient porter sur l'identification de la personne pour pouvoir rendre le système vraiment adaptatif à la personne présente et non pas à la présence d'une personne quelconque.

7.9.2 La localisation physique des individus et des services

Le développement d'applications utilisant Bluetooth pourra aider le système à prendre de meilleures décisions reliées à la position de l'usager par rapport aux appareils environnants. Les services qui utilisent Bluetooth peuvent aussi aider à identifier une personne géographiquement étant donné le court rayon du champ d'action d'une connexion Bluetooth (10m). Un service qui utilise Bluetooth est donc seulement disponible lorsque la

personne est à proximité de celui-ci. Présentement, la localisation peut se faire d'une manière paramétrique à l'aide de propriétés ou grâce à l'utilisation des fédérations qui représentent des pièces. De cette manière, il faut spécifier pour chaque service la pièce où il se situe. Les services Bluetooth peuvent par contre avoir une idée physique de leur position par rapport aux autres services offerts. Grâce à cette notion de proximité, on peut déplacer la logique de prise de décision face à la proximité des services. La logique de proximité qui est implantée de manière paramétrique peut se transformer pour être implanté de manière physique. Cette logique physique est possible étant donné qu'un service Bluetooth est seulement visible pour un autre service Bluetooth s'il est à moins de 10 mètres de celui-ci. Par le fait même, le service qui utilise Bluetooth sait qu'il est aussi à proximité de l'usager si ce dernier utilise un PDA Bluetooth. Une restriction possible est celle où deux services ou appareils se découvrent mais que ceux-ci sont situés dans deux pièces voisines qui ne seraient pas connectées directement par une porte ou un couloir. Il faudrait inclure un moyen pour que le système apprenne par lui-même ou par l'intervention d'un usager qu'un service est moins approprié qu'un autre à cause de cette éventualité.

La programmation Bluetooth est possible à ce jour, mais pas au niveau désiré pour ce projet-ci en utilisant le langage Java. Présentement, il faut programmer à partir d'un niveau très bas à cause des très grandes différences entre les implantations matérielles de Bluetooth par chacune des compagnies. L'uniformité du matériel Bluetooth n'est pas encore assez répandue pour pouvoir utiliser les API de Bluetooth définis par Java. Ces API sont très intéressants et de très haut niveau mais aucune implantation satisfaisante n'est disponible pour répondre au besoin de l'architecture Domus. On pourrait intégrer aussi les téléphones cellulaires qui sont de plus en plus répandus et qui supportent la technologie Java et Bluetooth.

7.9.3 Les alternatives à Jini

L'architecture UPnP[12] [16] permet le réseautage spontané et diffus de type client à client. Cette architecture utilise les protocoles TCP/IP, HTTP et le langage XML pour la description des services. Les appareils qui utilisent le standard UPnP peuvent se découvrir automatiquement et interagir entre eux. Les appareils UPnP peuvent aussi être intégrés à l'architecture Domus pour avoir une flexibilité encore plus grande. Lorsque les appareils UPnP sont physiquement connectés à un réseau, ils peuvent se connecter entre eux sans avoir besoin d'un serveur centralisé ou bien de l'intervention d'un usager pour la configuration de l'appareil. Le langage de programmation dans lequel ces appareils ont été conçus est sans importance puisque le protocole UPnP est basé sur TCP-IP pour la connexion au réseau et sur XML pour la description des services. L'intégration de services en utilisant la plateforme OSGI (Open Service Gateway Interface) [20] peut aussi être considérée. Avec l'intégration de SIP (Session Initiation Protocol) dans la plateforme OSGI, une multitude d'appareils et de services peuvent être accessibles depuis le réseau local d'une maison et aussi depuis l'extérieur (Internet).

[12] UPnP : L'acronyme UPnP ne veut pas dire dire **U**niversal **P**lug a**n**d **P**lay contrairement aux croyances populaire.

ANNEXES

A Pervasive Reminder System for Smart Homes

Sylvain GIROUX and Simon GUERTIN

Département d'informatique,
Université de Sherbrooke
2500 boul. Université, Sherbrooke, Canada J1K 2R1
{Sylvain.Giroux, Simon.Guertin,}@USherbrooke.ca
http://www.dmi.usherb.ca/~sgiroux/domus/

Abstract: One of the research projects at the DOMUS laboratory aims at the realization of smart homes and cognitive assistance for cognitively impaired people. The aimed system is a pervasive information system able to detect failure and anomalies in the behaviour of the habitants, and then to help them and notify other systems or people about potential danger. In this paper, we address the issue of finding the user and choosing the most appropriate device to transmit a message. As a prototype, a pervasive reminder system for message management explores the possibilities of dynamic cooperation between different kinds of devices, hardware, and software communicating through different networks. This pervasive system tries to locate, identify and notify the resident of upcoming appointments. The issues addressed are 1) localization and identification of the resident; 2) finding devices for communication and delivering messages according to the position of devices and the position of the user 3) spontaneous networking, discovery services, and service composition, and 4) migration of a running application. The latest allows a user working on a device to switch to another device more adapted to her task, making the application really pervasive.

1.1 Introduction

As Weiser said about pervasive computing, « The most profound technologies are those that disappear. They weave themselves into the fabric of everyday life until they are indistinguishable from it. » [1]. The DOMUS laboratory of the University of Sherbrooke studies pervasive computing and mobile computing. One of the research projects aims at the realization of smart homes and cognitive assistance for cognitively impaired people [2] [3]. People suffering from cognitive impairments —Alzheimer disease, schizophrenia, brain injuries...— often have no choice other than living in medical institutions. Smart houses [4] and mobile applications, e.g. activity compass [5], can play a central role in keeping them at home in their community. The aimed smart home would be a pervasive information system able to detect failure and anomalies in the behaviour of the habitants, and then to help them or to notify other systems or people about potential danger.

In this paper, we address the issue of finding the user and using the most appropriate device to transmit a message. As a prototype, a pervasive reminder system for message management explores the possibilities of dynamic cooperation between different kinds of devices, hardware, and software communicating through different networks. This pervasive system tries to locate, reach the user with the nearest or most convenient device and notify

her of upcoming appointments. To do that, we need at the implementation level to address spontaneous networking, discovery services, service composition, and migration of a running application

First we sketch the general issues raised by smart homes for cognitive assistance (§2). Then we describe the prototype: the physical set up (§3) and the functional requirements (§4). Once the goal and setting are clear, we detail the distributed implementation of the various components (§5).

2 General Issues

Let suppose Mrs Smith is suffering from the Alzheimer disease. She is cooking a stew and preparing coffee when somebody rings at the front door. She goes to answer, chats a bit, and forgets something is cooking. And she listens TV in the lounge. The smart home detects the situation is hazardous: nobody is in the kitchen and the stove is on for a long time. The system decides to send a reminder to her. So the issue to solve is first to find where she is, then find what devices in the lounge can be used to present the reminder — TV, telephone, her PDA…—, and finally package the message and send it to the chosen device. Hence to finally provide Mrs Smith with the appropriate feedback, a pervasive system must face many issues. In this paper, we explore the combination of the following ones:

- *Identification and localization of the appropriate user*: there may be many people simultaneously in the house; they can move from room to room.
- *Heterogeneity of devices, hardware, software and networks protocols*: in a house, there is a great variety of coexisting devices (electrical appliances, computers, TVs, PDAs…), software (server and client components, different operating systems and programming languages…), and networks (Ethernet, WiFi, Bluetooth, X10 [6]). They all have to collaborate.
- *Spontaneous networking and device and service discovery*: it is not possible to configure and to know beforehand all the devices and systems that will be present in the house; devices may join or leave the house dynamically; some are mobile, other are stationary.
- *Transitory coalition of devices*: devices and systems may offer complementary services that must collaborate for a moment to perform a task, for instance to deliver the message to a given user.
- *Pervasiveness*: interactions between components must be invisible and transparent to the user, furthermore a computation may have to migrate from one device to another. For instance, if the telephone is first used to reach Mrs Smith and she asks to see the state of the stove on TV, using the camera in the kitchen.
- *Zero-configuration*: obviously Mrs Smith cannot be asked to configure the system.

3 Prototype : a Pervasive Agenda System

The physical and software setup of the prototype incorporates multiple technologies that are not directly designed to work together (Fig. 1). In particular, the standard electrical hardware will be interconnecting with the computer hardware and software systems through the X10 protocol. A system may use different electrical devices to enhance the user's level of comfort or to extract information about different physical aspects of the house. X10

devices are affordable and are already made for home automation. The X10 devices used in the project are:

- one X10-computer interface device connects to the serial port of a computer on one end and into an electrical outlet on the other end. The device enables to read and write X10 codes on the electrical wires. Model: X10 TW523 Two-Way Interface Module.
- two wireless motion detectors. Model: Version II Wireless X10 Motion Sensor.
- one RF base to convert the signals from the wireless motion detectors into the electrical system. Model : X10 16 Device RF Base.

To read and write X10 codes from and to the computer interface, we use standard serial port programming in Java. Other devices used are

- a PC running Linux Red Hat 9 mostly used to run different services of the Domus infrastructure.
- the user's PC running windows XP in the main room.
- a standard wired LAN setup
- a wireless transmitter base to allow WiFi connections to the network.
- a standard home router with DHCP capabilities is used to interconnect the networked hardware and allow connections to the Internet. Model: Lynksys Cable/DSL 4-Port Router.
- a printer enhanced with a Bluetooth antenna to allow printing from diverse devices.
- a HP iPAQ Pocket PC h5550 with Bluetooth and WiFi capabilities.

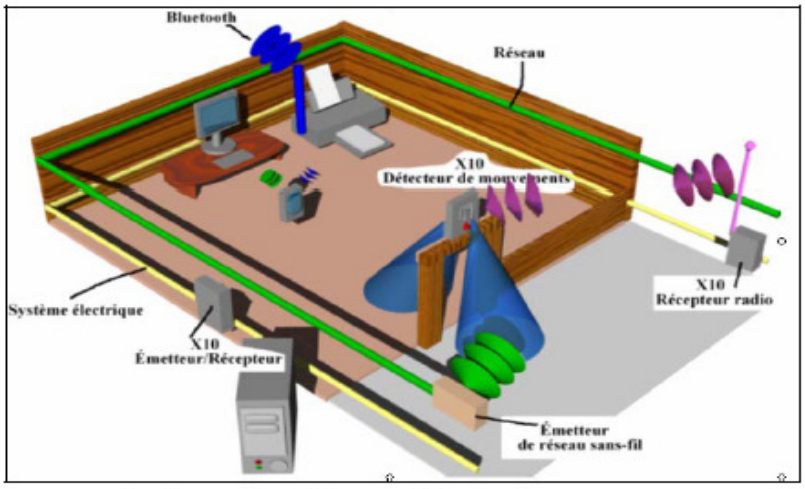

Figure 1 Current prototype implementation of the pervasive agenda system.

4 An Active and Pervasive Agenda System

The prototype built implements an appointment manager or agenda that keeps track of a user's appointments like any other agenda would do. Where the system distinguishes itself is that it is a distributed application that uses other distributed applications offering different services throughout the house. The agenda operates in conjunction with a messaging system to try to inform the user of an upcoming appointment. The system also uses other features from the house to better accommodate the user's needs.

To perform its duties, the system first must search for a device to notify the user (§4.1), then localize the user and prepare the delivery of the message to the user (§4.2), and

finally support the migration of a user session from device to device to follow a user from room to room (§4.3). The next sections will present how these issues were addressed in the prototype.

4.1 Search of Devices for Message Delivery

When an appointment is approaching, the system has to notify the user. To do this, the system attempts to use any service implementing the messaging service. The system might or might not know where the user is in the house, so it disseminates the message to any resources that may communicate the information to the user. When the user reads the message, the system gets a confirmation the user got the message. The system then stops trying to reach the user and notifies other messaging services in the house so they can remove the read message.

4.2 Preparing Message Delivery to a Nearby User

When the system knows a certain user is nearby, it prepares to log the user in. Due to limited means of user identification in the current setup, we consider only one user. In this scenario, motion detectors work in collaboration with other software services running in the Domus infrastructure. The agenda is usually in standby mode in the main room. When the user enters the room as detected by motion detectors, the agenda login screen is filled with user name. If the user wants to use the application, she just needs to type in her password and start using the tool. The agenda program can do this because it uses a service that informs any client about the presence of a user in the room. When the user leaves the room, the agenda is also informed and reacts accordingly by removing the pre-filled login.

4.3 Migrating user session from device to device

A user may need to migrate from one device to another when working with an application (in our case in the agenda). She may need better display capabilities. She may prefer to work in another room. She may need to work on a wireless device. When the user switches device, she wants to pick up the session where she left it. There are three cases when this happens:

- the user directly switches to the other device if it is available;
- the user stores her session on the server if no available device meets her needs;
- the user session is retrieved from the server when the device becomes available.

In the first two cases, the agenda system will log off the user from the current device when the transfer is successful. A login screen with the user name already filled in will be displayed to the user on the other device. The session will then be restored to where it was left on the previous device. In the second case, the user is logged off from the current device when the session is transferred to the server successfully. The user will then have to log on the appropriate device when it becomes available (third case). The session can then be restored from the server from the agenda's menu commands (or it could be automatically restored on a successful login).

5 Implementation

In this section we present implementation details of the prototype. First we explained the choices underlying the general infrastructure (§5.1). Next we present general services that must run in any cases (§5.2). Then we present specific services: services related to X10 components (§5.3) and services related to message delivery (§5.4). Finally the agenda system as a whole is described (§5.5).

5.1 Jini and Java as Backbone for the General Infrastructure

Indeed we want to build dynamic distributed system on heterogeneous components and supporting code migration. Hence we choose Java[1] and Jini. Java provided the compulsory multi-platform support and code migration through RMI. Jini provided

- *spontaneous networking*: services offered to the users or to other services may join and leave anytime.
- *service registration and discovery*: Jini discovery services enable to find the needed services in the DOMUS federation.

Reggie[2] is running on the Linux PC in a persistent and activable mode to save CPU resources. An http server is running also on this computer to make the necessary Jini jar files available for download on the house's LAN.

There is no centralized system but many different kinds of services that can have different physical or logical location. Services are grouped in the main federation: DOMUS. In fact, many sub-federations are used to realize the whole system (Fig. 2). A federation could either represent a room, a floor, the whole house or any logical boundary. With a good use of federations, a service or client can better choose a specific service depending where the desired service is registered, giving hints on its location.

5.2 General Services

For the Domus infrastructure, we developed a Domus server. This server is a service that provides basic information about this infrastructure. It also provide another HTTP server that allows to upload files that other services might need to store and make available to other programs that will use the system. Those files contain the code to distributed for a Jini service to work. These files need to be available through an HTTP server. This feature simplifies the development of new services.

The security server is a service that is used by any other servers or client program that needs to authenticate a user. This service also runs on the Linux PC. It is a small database that stores usernames and passwords.

5.3 Services related to X10 Components

The X10 server allows the X10 technology to be incorporated in the Domus infrastructure. This server uses serial port programming in Java to interact with the X10 Computer Interface (§3). This computer interface monitors X10 codes travelling on the electrical

[1] We used JDK1.4 with the latest release of Jini. We needed to downgrade the version of Java we used when we incorporate the iPAQ because J2ME is only compatible with the JDK1.3.
[2] Reggie is the lookup service used in Jini to allow services to register themselves and lookup other services.

wires and stores them in the device's memory. With the X10 server, other services can read or write those codes to know the state of devices or to control them. To use it, a client or service of the Domus federation registers to the X10 server and specifies which codes it is interested in (Fig. 3.).

The room's motion detector and darkness detector service allows other services to show a more adapted behaviour in the presence of a user. This service is available to any other client or service interested in the state of occupation of the room or in the state of lighting. This service is composed of

- the two motion detectors that are placed back to back above the room's entrance;
- the wireless base that transmit the signals of the two wireless detector's X10 into the electric circuit of the house;
- the X10 server;
- the motion detector program that will handle the X10 signals.

The motion detector program registers to the X10 server and specifies X10 codes it needs from the motion detectors. It will then analyze the X10 codes received from the X10 server to determine either the room is empty or not, in full darkness or not. The algorithm can detect leaving or entering the room by analyzing the time elapsed between the X10 codes thrown by each motion detectors. Other services can then register themselves to the motion detector service to be informed of the room's occupation status.

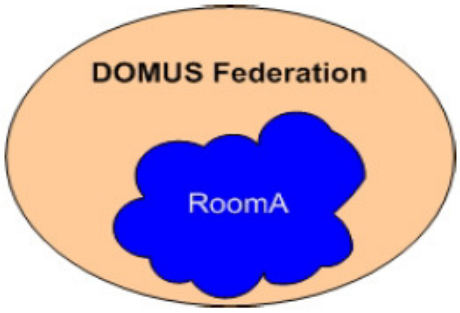

Figure 2 Organizing Domus into sub-federations.

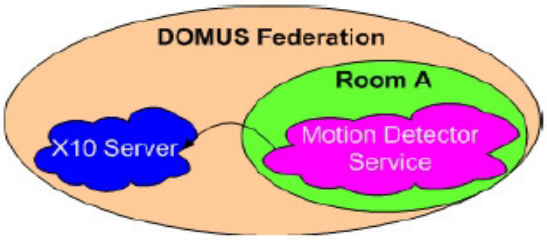

Figure 3 Detecting presence with X10 components.

5.4 Messaging System Server

To deliver messages to users in the house, we implemented a message server. This message server accepts request from clients that wants to deliver a message to a specific recipient. The message server processes the requests and looks up all the message clients in the Domus federation to deliver the message. The server can handle new message clients joining the federation. When a message has been read and confirmed by the recipient, the message client informs the message server of the successful delivery. The message server then proceeds to inform the originator of that message of the successful delivery. The message server also has to inform all the other message clients to remove this specific message from their queue.

Tasks of message clients are simpler. They have to display the message any way they want but they must keep a certain degree of confidentiality and integrity. The message client receives message request from the message server and then waits for a user

interaction. For a user to read a message, she normally enters her password for confirmation. The message client confirms the user's identity with the security server. After successful user authentication, the message client then contacts the message server to acknowledge a successful delivery. Message clients must have the basic functions of adding a message and removing a message when requested by the server.

5.5 The Agenda System

The agenda server is a service that handles appointments for the users and tries to notify them of an upcoming appointment. The server has all the basic functionality an agenda server would have. That is to create, store, retrieve, and delete appointments. It uses the security server to authenticate users upon connection of clients. What makes the agenda server unusual is when time comes to notify a user. It searches the Domus federation for messaging services able to send out the notification to the right recipient (Fig. 4). Once the server has received a confirmation of reception by the messenger service it stops and marks the appointment as acknowledged by the user. The server also takes care of overdue appointments that were not acknowledged and the server removes them from the messenger service.

The server also supports the storage and retrieval of a user's session if she wishes to switch device. At the beginning of a session, the user gives a user name and password. She then uses the agenda like a normal agenda to browse, add and delete appointments. If she wants to switch from one computer to another that better suits her needs, she can browse the available computers and directly migrate her session to the new computer if it is available. If no computer is available, the user can store her session to the agenda server to retrieve it later from any other computer when available. When switching directly to another computer, a login screen with the user name is automatically pre-filled. The user enters her password and the session is restored as it was left on the previous computer. (Fig. 5.). There is no need to reopen windows or retype the information, even if not saved.

In the project we assume that it is always the same person that enters or leaves the room. When the client is started, it is looking for the motion detector service to be informed of the presence of a user in the room. There might be multiple motion detector services in the whole system, but the client will only be interested in a motion detection service that belongs to the same Jini federation as the agenda service. This permits the agenda to start the login process when someone enters the room. If there is no motion detector service for the room, the agenda client just behaves normally without inferring somebody is in the room. The client is also informed of the lighting of the room through the motion detector service. It could then open up a light or any other service that would enhance the environment for the user if she chooses to use the agenda (this is not yet implemented).

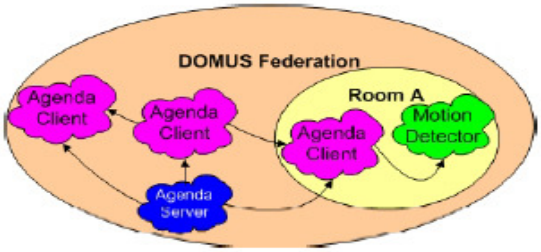

Figure 4 Agenda clients and server.

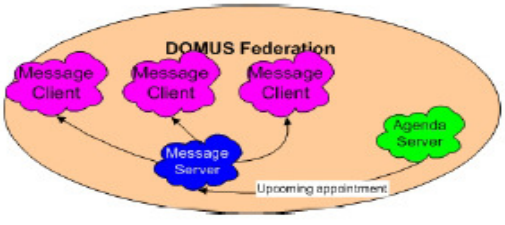

Figure 5 Request from the server to notify a user.

6 Conclusion

Smart homes providing cognitive assistance for cognitive impaired people raise many difficult issues. In the long-term, the DOMUS laboratory wants to implement such a smart home as a pervasive information system able to first detect failure and anomalies in the behaviour of the patients, and then to help them on-site and to perform some sort of tele-monitoring. Because it is a very complex task, we decided to address a simplified version of this problem.

In this paper we thus described the implementation of a pervasive distributed agenda system. This system has been used to explore how sensing and information devices can be integrated using a variety of communications services on the one hand and how information can be delivered to the user in a pervasive environment on the other hand. The emphasis was put on locating and communicating information to residents. The prototype enabled to explore the possibilities of dynamic cooperation between heterogeneous devices, hardware, and software communicating through different networks. This resulting pervasive system tries to locate, identify and notify the resident of upcoming appointments. Among the issues that have been addressed, let stress: 1) localization and identification of the resident; 2) finding devices for communication and delivering messages according to the position of devices and the position of the user 3) spontaneous networking, discovery services, and service composition, and 4) migration of a running application. The latest allows a user working on a device to switch to another device more adapted to her task. The user can then go on with the application without losing her work, making the system really pervasive. This distributed system is adaptive to changes of availability of services, devices and computers.

We intend to use the agenda system as a reminder system in an Agenda Home [7]. Robustness will then be a key issue to address. With respect to monitoring in a smart home, we need a way to store the vital information in case the network or a service is down. When the service or the network would become available again, the services would then report back the information that was saved locally.

Finally we intend to exploit the limited range of Bluetooth to build local coalition of devices based on physical proximity.

7 References

[1] Mark Weiser, *The Computer for the 21st Century*, Scientific American, 1991.

[2] H. Pigot, et al., *The intelligent habitat and everyday life activity support. 5th international conference on Simulations in Biomedicine*, Slovenia, April 2003.

[3] Pigot H., et al., *The role of intelligent habitats in upholding elders in residence. 5th Intl Conf. on Simulations in Biomedicine*, 2003, Slovenia. WIT Press

[4] Rialle .V, et al. *An experimental Health Smart Home and its distributed Internet-based Information and Communication System. MEDINFO 2001,* IOPress. pp. 1479-1483.

[5] Kautz, H. et al. *An Overview of the Assisted Cognition Project*, AAAI 2002, workshop.

[6] Boone, K, *Using X10 for Home Automation*, http://www.kevinboone.com/x10.html

[7] Pigot, H., and S. Giroux, *Keeping in Touch with Cognitively Impaired People: How Mobile Devices Can Improve Medical and Cognitive Supervision*, 2nd International Conference on Smart Homes and Health Telematics ICOST 2004, Sept 15-17, 2004, Singapore.

BIBLIOGRAPHIE

[1] Architecture Jini Surrogate, version 0.94, 2001: http://surrogate.jini.org/overview.pdf

[2] C. Bala Kumar, Paul Kline et Tim Thompson, **"Bluetooth Application Programming With the Java Apis"**, Morgan Kaufmann Publishers, 2003.

[3] Domenico Cotroneo, Almerindo Graziano, Stefano Russo, **"Security Requirements in Service Oriented Architectures for Ubiquitous Computing"**, *Proceedings of the 2nd workshop on Middleware for pervasive and ad-hoc computing*, ACM Press, 2004, pp. 172-177.

[4] N. Davies, S.P. Wade, A. Friday et G.S. Blair, "**Limbo: A tuple space based platform for adaptive mobile applications**", *Proceedings of the International Conference on Open Distributed Processing/Distributed Platforms (ICODP/ICDP '97)*, Toronto, Canada, 27-30 Mai 1997, pp. 291-302.

[5] Kevin Eustice, Leonard Kleinrock, Shane Markstrum, Gerald Popek, V. Ramakrishna, Peter Reiher, **"Securing nomads: the case for quarantine, examination, and decontamination",** *New Security Paradigms Workshop, Proceedings of the 2003 workshop on New security paradigms*, ACM Press, 2003, pp. 123-128.

[6] Innes A. Ferguson et James D. Davlouros, "**PeopleFinder: A Multimodal Multimedia Communications Tool for Interconnecting Network Users**", In *Working Notes of the IJCAI-95 Workshop on Intelligent Multimedia Information Retrieval*, Montreal, PQ, Août 1995, pp. 2059-2060.

[7] Robert Grimm, Janet Davis, Ben Hendrickson, Eric Lemar, Adam MacBeth, Steven Swanson, Tom Anderson, Brian Bershad, Gaetano Borriello, Steven Gribble, David Wetherall **"Systems Directions for Pervasive Computing"**, Proc. 8th Workshop Hot Topics in Operating Systems (HotOS VIII), 2001, pp. 128-132.

[8] Bruce Hopkins, Ranjith Antony et John Zukowski, **"Bluetooth for Java"**, APress L P., 2003.

[9] B. Johanson et A. Fox, **"Tuplespaces as Coordination Infrastructure for Interactive Workspaces"**, *Proc. Workshop Application Models and Programming Tools for*

Ubiquitous Computing (Ubitools 01), 2001,
http://graphics.stanford.edu/papers/eheap_ubitools01/eheap-posn-paper.pdf
[10] Tim Kindberg et Armando Fox, "**System Software for Ubiquitous Computing**", *IEEE Pervasive Computing*, vol. 1, no. 1, Jan. 2002, pp. 70-81.
[11] Shankar R. Ponnekanti, Brian Lee, Armando Fox et Pat Hanrahan and Terry Winograd, "**ICrafter: A Service Framework for Ubiquitous Computing Environments**", *Proceedings of UBICOMP 2001*, 2001, pp. 56-75.
[12] Larry Press, "**Personal Computing: The Post-PC Era**", *ACM Press*, vol. 42, no. 10, Oct. 1999, pp. 21-24.
[13] Tristan Richardson, Quentin Stafford-Fraser, Kenneth R. Wood et Andy Hopper, "**Virtual Network Computing**", *IEEE Internet Computing*, vol. 2, no. 1, Jan.-Fév. 1998, pp. 33-38.
[14] Specification IP Interconnect, version 1.0 Draft Standard 3:
http://ipsurrogate.jini.org/sa-ip.pdf
[15] Vince Stanford, "**Using Pervasive Computing to Deliver Elder Care**", *IEEE Pervasive Computing*, vol. 1, no. 1, Jan. 2002, pp. 10-13.
[16] UPnP™ Implementers Corporation:
http://www.upnp-ic.org/docs/UPnP%20Q-A%205-16-2003.pdf
[17] Jim Waldo, **"The Jini architecture for network-centric computing"**, *Communications of the ACM*, vol. 42, no. 7, Jul. 1999, pp. 76-82.
[18] Jim Waldo, **"The Jini Specifications, Edited by Ken Arnold (2nd Edition)"**, Addison-Wesley Professional, 2000.
[19] Mark Weiser, **"The Computer for the 21st Century"**, *Scientific American*, vol. 265, no. 3, Sept. 1991, pp. 94-104.
[20] Daqing Zhang, Zhengning Dai, Xiaohang Wang, Xiao Ni et Song Zheng, **"A New Service Delivery and Provisioning Architecture for Home Appliances"**, *Toward a Human-Friendly Assistive Environment*, IOS Press, 2004, pp. 221-228.
[21] E. Gabber et A. Wool, **"How to Prove Where You Are"**, *Proc 5th ACM Conf. Computer and Comm. Security*, ACM Press, 1998, pp. 142-149.

[22] T. Kindberg et P. Nikander, **"Context Authentication Using Constrained Channels"**, *tech. report HPL-2001-84, Hewlett-Packard Laboratories,* Palo Alto, Calif., 2001.