Une Infrastructure à Composants pour des Applications Ubiquitaires

Areski Flissi, Christophe Gransart, Philippe Merle

Une Infrastructure à Composants pour des Applications Ubiquitaires

Areski Flissi
LIFL / CNRS
Université des Sciences et Technologies de Lille (USTL)
59655 Villeneuve d'Ascq, France
Areski.Flissi@lifl.fr

Christophe Gransart
INRETS-LEOST
20 rue Elisee Reclus, BP 317
59666 Villeneuve d'Ascq, France
Christophe.Gransart@inrets.fr

Philippe Merle
INRIA-Futurs / LIFL
Université des Sciences et Technologies de Lille (USTL)
59655 Villeneuve d'Ascq, France
Philippe.Merle@inria.fr

RESUME
La multiplication des terminaux mobiles et la généralisation des réseaux sans fil impliquent des changements dans la conception et l'exécution des applications logicielles. La problématique est maintenant clairement identifiée sous le terme informatique ubiquitaire. Nous présentons dans ce papier une infrastructure dédiée aux services contextuels ubiquitaires. Ceux-ci sont conçus sous la forme d'assemblages de composants logiciels distribués et découverts dynamiquement en fonction de la localisation du terminal utilisateur et de ses caractéristiques, puis automatiquement déployés sur ce dernier. Nous avons implanté cette infrastructure, ainsi qu'un exemple de service, au dessus du modèle de composants CORBA de l'OMG et de la plate-forme libre OpenCCM.

Mots clefs
Informatique ubiquitaire, composants et objets distribués mobiles, intergiciel, CCM.

ABSTRACT
Multiplication of mobile devices and generalized use of wireless networks imply changes on software applications design and execution. This new challenge is now called ubiquitous computing. In this paper, we present a software infrastructure dedicated to ubiquitous contextual services. They are designed as assemblies of distributed software components and are dynamically discovered according to end-users' physical location and device capabilities and automatically deployed on it. We have implemented this software infrastructure and an example of service on top of the OMG CORBA Components Model and the OpenCCM open source platform.

Categories and Subject Descriptors
C.2.4 [Computer-Communication Networks]: Distributed Systems

General Terms
Design, Experimentation.

Keywords
Ubiquitous computing, distributed mobile components and objects, middleware, CCM.

UbiMob'05, May 31 - June 3, 2005, Grenoble, France.

1. INTRODUCTION
La multiplication des terminaux mobiles et la généralisation des réseaux sans fil nécessitent des changements dans la manière dont les applications logicielles sont conçues et s'exécutent. Ce nouveau challenge [14], appelé informatique ubiquitaire, soulève de nombreux problèmes : hétérogénéité des équipements, ressources limitées, distribution des applications, mobilité des terminaux, sécurité, découverte des services, déploiement automatique du logiciel sur le terminal, etc. Il n'existe actuellement pas de solution générale qui couvre le cycle complet du processus de construction des applications ubiquitaires, de la conception à l'exécution. Nous proposons dans cet article une approche globale à base de composants logiciels pour la conception, la découverte dynamique, le déploiement automatique et l'exécution de ces services pour usagers mobiles[1]. Les problèmes relatifs à la sécurité et à la gestion de la déconnexion ne sont pas discutés ici. Dans notre approche, un service est modélisé sous la forme d'un assemblage de composants logiciels distribués. Une infrastructure, elle-même à base de composants, permet aux utilisateurs de découvrir dynamiquement les services adaptés à leur terminal et de déployer automatiquement la partie du logiciel s'exécutant sur leur terminal. Les interactions entre les différents composants distribués sont réalisées via l'utilisation d'un intergiciel.

Cet article est organisé de la manière suivante. La section 2 décrit à l'aide d'un exemple dans le domaine des transports, les problèmes à résoudre dans le contexte de l'informatique ubiquitaire. La section 3 présente le modèle de conception de services. La section 4 détaille l'infrastructure à composants pour la découverte, le déploiement automatique de services contextuels, et présente une implantation avec le modèle de Composants CORBA [11] et la plate-forme OpenCCM [10]. La section 5 discute des travaux relatifs et la section 6 conclut sur notre contribution.

2. CONTEXTE
Cette section traite au travers d'un exemple fil rouge des problèmes relatifs aux applications ubiquitaires.

2.1 Un Exemple de Service Contextuel
Nous illustrons notre propos à l'aide d'un exemple simple dans le domaine ferroviaire. Un navetteur muni d'un assistant numérique Wi-Fi arrive dans une gare pour prendre un train. Lorsqu'il se

[1] Ce travail est réalisé dans le cadre du projet CPER-TAC MOSAIQUES, avec le soutien du Conseil Régional de la Région Nord Pas de Calais et de la Direction Régionale à la Recherche et à la Technologie.

trouve dans le hall, son PDA détecte automatiquement le lieu et lance un service fournissant des informations sur le quai de départ du train, les horaires, etc. Ce service peut être conçu avec différentes interfaces utilisateur (voir Figure 1) : graphique ou sonore avec synthèse vocale.

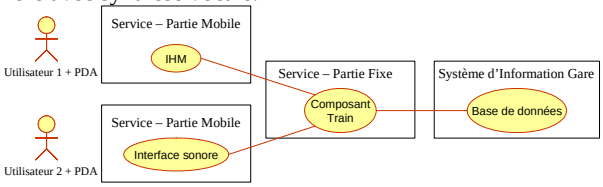

Figure 1 : Cas d'utilisation du service.

2.2 Problèmes à résoudre

Pour réaliser ce scénario assez simple, nous devons prendre en compte un certain nombre de difficultés inhérentes aux applications contextuelles et ubiquitaires :

- Hétérogénéité des équipements

Les équipements utilisés sont très variés : un utilisateur peut se servir d'un ordinateur portable, d'un PDA ou encore d'un *smart phone*. Ces équipements fonctionnent avec des systèmes d'exploitation variés (tels que Windows, Linux, Windows CE, PalmOS, Symbian, etc.). Du point de vue du réseau sans fil, différentes technologies sont à la disposition de l'utilisateur : Wi-Fi (IEEE 802.11a/b/g) ou Bluetooth pour des communications courtes/moyennes portée, GPRS ou UMTS pour une couverture nationale voire internationale.

- Ressources limitées

La consommation énergétique est un problème rencontré par tous les équipements mobiles. De plus, les PDA et les *smart phones* disposent de peu de mémoire et d'une puissance limitée.

- Applications distribuées

Notre exemple fil rouge qui doit permettre à un usager d'obtenir des informations sur son train est réparti sur plusieurs machines. L'interface graphique s'exécute sur le PDA du navetteur alors que le traitement concernant l'extraction des données s'effectue sur des machines installées à demeure dans la gare. La communication entre les parties fixes et mobiles se fait via un réseau sans fil. La réalisation concrète de ce service doit utiliser un intergiciel de communication. Différentes technologies sont disponibles telles que CORBA, les Web Services, un système de messagerie asynchrone, etc.

- Découverte de services

Il faut permettre à un utilisateur de savoir que des services sont disponibles en un lieu précis, en fonction de la géo-localisation. De plus, le code des services disponibles à déployer sur son terminal est différent selon qu'il utilise par exemple un PDA noir et blanc ou un ordinateur portable couleur. Pour faire ce choix, le système doit avoir des informations sur les caractéristiques du terminal afin de présenter à l'utilisateur un sous-ensemble de services qui potentiellement peuvent s'exécuter sur son terminal.

- Déploiement de services

Dès qu'un utilisateur est intéressé par un service, l'infrastructure doit être capable de déployer le code de ce service sur son terminal. Plus précisément, la partie cliente du logiciel implantant le service (l'IHM affichant la liste des trains au départ) est téléchargée puis instanciée sur le terminal à la demande de l'utilisateur. La partie fixe du service (le système d'information de la gare et le code du traitement pour l'extraction et l'envoi des données) est supposée avoir été déployée préalablement.

3. NOTRE MODELE DE CONCEPTION

Afin de relever les défis précédents, nous avons choisi de concevoir les services contextuels ubiquitaires selon une approche à composants. Un service est constitué d'un ensemble d'assemblages de composants logiciels distribués et interconnectés. Nous distinguons deux types d'assemblages : les fixes et les mobiles. L'assemblage fixe représente la partie permanente d'un service : il est déployé sur les machines fixes de l'environnement ubiquitaire lors la mise en route du service et est présent tant que le service est disponible. L'assemblage mobile représente la partie du service dédié à un utilisateur : il est déployé automatiquement à la demande de l'utilisateur et est automatiquement détruit lorsque celui-ci quitte la zone de couverture contextuelle. Chaque composant possède un ensemble d'attributs configurables et des ports identifiant les fonctionnalités offertes ou requises par le composant. Nous réutilisons les quatre types de ports définis par le modèle de composants CORBA : les facettes, les réceptacles, les puits et les sources d'événements. Une facette et un réceptacle exposent une interface, *i.e.* un ensemble de méthodes, respectivement implantée ou requise par le composant. Un puit et une source identifient qu'un composant peut respectivement consommer ou produire un certain type d'événements. La connexion d'un réceptacle vers une facette et d'un puit sur une source permet la mise en place des interactions respectivement synchrones et asynchrones entre composants.

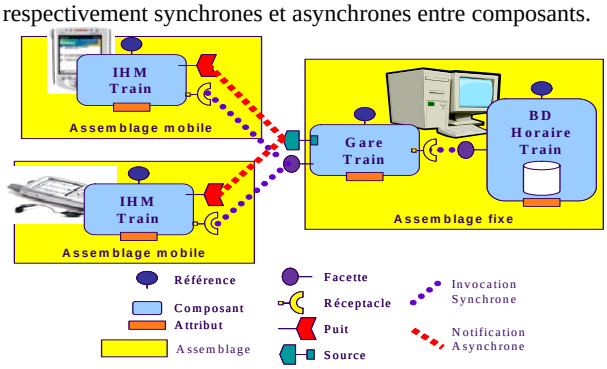

Figure 2. L'assemblage des composants du service "Trains".

La figure 2 résume l'ensemble de ces concepts avec le service « Trains » sous la forme d'un assemblage fixe de deux composants permanents et de plusieurs assemblages mobiles constitués chacun d'un composant « IHM Train ». Le composant « IHM Train » est déployé sur le terminal de l'utilisateur et est interconnecté avec le composant fixe « Gare Train » afin d'obtenir les informations sur les trains de la gare. Il doit être implanté différemment selon le type de terminal (PDA avec un écran de petite taille, ordinateur portable ou téléphone mobile). Chaque composant est constitué de plusieurs codes d'implantation et d'une description de leurs dépendances. Cette dernière identifie le contexte adéquat en termes de ressources matérielles et logicielles.

4. NOTRE INFRASTRUCTURE

Nous avons fait le choix de modéliser notre infrastructure suivant une approche à base de composants logiciels. Les composants sont responsables de la découverte des services contextuels, du calcul des assemblages adaptés aux terminaux et de leur déploiement sur le terminal utilisateur.

4.1 Découverte Dynamique de Services

La découverte de services est un élément clé de l'informatique

ubiquitaire. Le défi consiste à informer un utilisateur entrant dans une zone précise de l'existence de services. Pour cela, nous introduisons dans un premier temps un composant que nous nommons *Service Registry* (SR), hébergé sur une machine fixe de l'environnement ubiquitaire (par exemple un serveur de la gare). Le composant SR est chargé de gérer la liste des services (ajout/mise à jour/suppression). Il transmet un identifiant unique, qui permet de différencier les fournisseurs de services. En effet, du point de vue de l'utilisateur, plusieurs environnements ubiquitaires peuvent co-exister. Dans notre exemple, il peut s'agir d'un musée proche proposant des informations sur les expositions en cours. Le SR calcule également l'assemblage mobile du service adapté au terminal, en fonction des capacités matérielles et logicielles de ce dernier et est responsable de l'envoi de la liste des services disponibles.

Dans un second temps, un composant, nommé *Service Activator* (SA) et démarré sur les terminaux utilisateurs, est à l'écoute des ID transmis par chaque SR. A partir de cet ID, le composant SA est en mesure d'interroger le SR afin de connaître les services disponibles. Le composant SA transmet ainsi les caractéristiques matérielles et logicielles du terminal au SR sélectionné. Enfin, il reçoit et affiche sur le terminal utilisateur les services adaptés.

Grâce à cette architecture, le problème de la découverte de services se ramène à celle du composant SR par les SA. Deux scénarios sont possibles en fonction du tiers qui initie le processus : le terminal de l'usager ou le serveur fixe hébergeant le SR. Dans le premier cas, le terminal mobile émet périodiquement des requêtes (mode transmission, TX). Dans le second cas, le terminal de l'utilisateur est en mode réception (RX). La diffusion des requêtes périodiques à travers le réseau signalant la présence d'un SR est à l'initiative du serveur. Ce second cas est très intéressant si on prend en compte plusieurs paramètres. Premièrement, du point de vue énergétique, la Table 1 montre qu'un équipement en mode réception consomme en moyenne 30% de moins qu'en mode émission Deuxièmement, avoir un seul émetteur (le serveur fournissant les services) et plusieurs utilisateurs (les terminaux) génère moins de trafic que l'architecture opposée. Ceci est d'autant plus important que le nombre de terminaux impliqués est élevé. Finalement, du point de vue de l'utilisateur, ce mode de fonctionnement permet une découverte transparente des services et les mises à jour sont automatiquement et immédiatement notifiées aux terminaux.

Table 1. Consommation énergétique de cartes Wi-Fi.

	TX	RX	RX/TX
Compaq WL110 [5]	280 mA	180 mA	65 %
Buffalo AirStation G54 [4]	550 mA	350 mA	64 %
Dlink DWL-AG660 [6]	500 mA	379 mA	75 %

Une fois le SR découvert par le terminal, le SA émet une requête et obtient la liste des services adaptés à l'équipement. Pour cela, nous avons défini un protocole de « négociation » entre le terminal mobile et le serveur du fournisseur de services (Figure 3). Après la découverte du SR (1), les caractéristiques du terminal sont envoyées au SR (2). Celui-ci calcule alors les assemblages appropriés. Plusieurs implantations des composants constituant la partie cliente du service sont possibles. Pour l'exemple fil rouge, il faut pouvoir gérer des interfaces hétérogènes, par exemple. Plusieurs configurations d'assemblage peuvent également être proposées, comme un composant *Traducteur* qui serait intercalé entre les composants *IHM* et *Trains*. La liste envoyée à l'utilisateur comporte uniquement les services contextuels adaptés à son terminal (3).

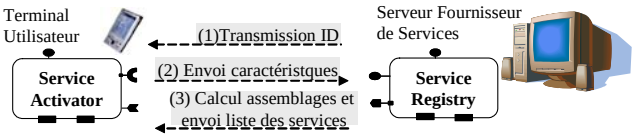

Figure 3. Découverte des services et protocole de négociation.

4.2 Déploiement Automatique et Exécution

Une fois un service découvert et sélectionné par l'utilisateur, l'infrastructure doit déployer le code sur le terminal mobile. Ceci consiste à télécharger les binaires des composants depuis un référentiel vers le terminal, instancier les composants et les configurer (*e.g.* création de l'IHM) et enfin les interconnecter avec les composants de la partie fixe du service (le système d'information de la gare). Pour ce faire, nous introduisons un troisième composant nommé *Deployer* (DP). DP est préalablement déployé sur les terminaux mobiles, comme pour le composant SA.

La figure 4 complète l'architecture de notre infrastructure. Elle illustre les étapes du choix du service par l'utilisateur (4) jusqu'au déploiement de l'assemblage mobile du service (5, 6a et 6b), c'est-à-dire la partie cliente de l'application. La partie fixe du service a été déployée préalablement suivant le même processus. En ce qui concerne le référentiel d'assemblages, celui-ci peut être hébergé par le serveur du fournisseur de services ou bien sur un site distant accessible via HTTP ou FTP par exemple. Dès la fin de ce processus, l'utilisateur final peut accéder aux fonctionnalités du service.

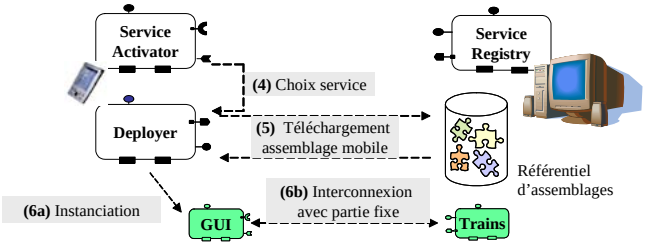

Figure 4. Choix du service et déploiement sur le terminal.

Le rôle du composant DP de l'infrastructure ne se limite pas au déploiement des assemblages mobiles. En effet, il gère également le cycle de vie de l'application dans un contexte ubiquitaire où les utilisateurs/terminaux entrent et sortent de la zone de couverture réseau. Imaginons le scénario suivant : l'utilisateur consulte les informations sur son train et monte dedans. Lors de sa prochaine visite, il aimerait profiter immédiatement du service « Train », sans passer par les étapes de découverte du SR, de transmission des caractéristiques, de négociation, du choix du service, etc. L'idée est, au moment de quitter la zone de couverture (le train démarre), de proposer de garder en cache l'assemblage mobile afin de n'avoir qu'à le réinstancier ultérieurement (étapes 6a et 6b). La gestion des versions des services est réalisée en affectant un identifiant unique à chaque service et version.

4.3 Expérimentation

Nous avons implanté notre infrastructure pour les applications ubiquitaires avec le modèle de composants CORBA et l'implantation libre OpenCCM écrite en Java. Chacun des composants de l'infrastructure (SA, SR, DP) est implanté par un composant CORBA. De plus, un composant générique *Multicast* a été conçu pour réaliser les communications *multicast* (envoi et écoute) entre les composants SR et SA lors de la phase de

découverte des services.

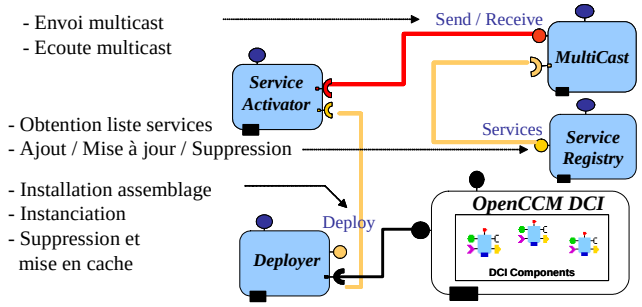

Figure 5. L'assemblage de composants CORBA.

L'utilisation de requêtes *multicast* est bien adaptée ici car cela permet aux composants de l'infrastructure hébergés sur les terminaux utilisateurs (SA et DP) de n'être, d'un point de vue CCM, pas « connectés » aux composants hébergés sur le serveur du fournisseur de services (SR). Ce dernier point est crucial car le terminal de l'utilisateur n'est pas censé connaître son environnement avant de le découvrir ! La figure 5 représente l'assemblage de composants CORBA de l'infrastructure connecté aux composants de la plate-forme OpenCCM. Concernant le déploiement, nous avons utilisé directement les interfaces offertes par l'infrastructure OpenCCM DCI [3] pour le téléchargement des assemblages, les instancier, les configurer puis les détruire.

5. TRAVAUX SIMILAIRES

Le courtier OMG/ISO *trader* [12] et le *trader* RM-ODP [9] gèrent la publication et la découverte de services. Notre composant SR peut être implémenté à l'aide de ces technologies. JINI [7], développé par Sun Microsystems, offre différentes manières de découvrir les services. Le protocole *Multicast Request Protocol* permet à un terminal usager d'envoyer des requêtes *multicast* pour trouver un serveur. La réponse est retournée au terminal usager sous forme d'un message TCP. La solution opposée, appelée *Multicast Announcement Protocol* est utilisée par les services de recherche pour annoncer leur présence aux différents terminaux. Ce mécanisme est similaire à celui que nous avons implémenté.

Le produit *Appear Provisioning Server* [1] de la société AppearNetworks est une infrastructure pour la découverte et le déploiement de services. Le produit permet de déployer sur des terminaux mobiles des applications ou des documents lorsque les usagers sont en des lieux précis. Les applications sont développées indépendamment de la plate-forme de diffusion. Dans ce sens, ils ne fournissent pas de solution sans couture pour l'infrastructure et les services.

Il existe de nombreux travaux de recherche autour des infrastructures pour l'informatique ubiquitaire [8]. En se limitant aux infrastructures à base de composants CORBA, le projet AMPROS [2] étudie et développe un intergiciel pour environnement mobile. Comme dans notre proposition, les travaux se basent sur la plate-forme OpenCCM. Un ensemble d'outils est développé pour permettre l'adaptation des applications à l'exécution. Enfin, les travaux de [13] sur le modèle *Satin* visent à fournir des assemblages de composants dotés de propriétés de sûreté. Ces travaux sont complémentaires à notre plate-forme.

6. CONCLUSION ET PERSPECTIVES

Partant du constat qu'il n'existe pas actuellement de solution globale pour les applications ubiquitaires, nous avons présenté un modèle de conception à base de composants logiciels distribués, avec une infrastructure pour l'exécution de services contextuels ubiquitaires. Cette solution traite entre autres les problèmes relatifs à la découverte et au déploiement automatique sur les terminaux utilisateurs. et a été implantée au dessus du modèle de composants CORBA et de la plate-forme OpenCCM.

Ce travail ouvre plusieurs perspectives. Dans un premier temps, nous souhaitons nous concentrer sur la partie « négociation » entre les terminaux et la partie fixe de l'infrastructure, et plus précisément le calcul des assemblages adaptés aux terminaux en fonction de leurs caractéristiques, en nous basant entre autres sur les travaux existants. Un autre objectif est d'améliorer la plate-forme OpenCCM afin de pallier certaines limitations : les communications *multicast* et les *threads* applicatifs doivent être codés explicitement dans les composants ; la sécurité et la gestion des déconnexions ne sont pas prises en charge. Cela nécessite d'étendre les conteneurs OpenCCM pour gérer ces propriétés non-fonctionnelles. De plus, des travaux sur l'amélioration des temps de chargement des archives vont être menés. Ceci permettra de prendre en compte la possible faiblesse de débit du réseau sans fil et la taille des archives des composants. Nous pensons nous orienter vers une génération dynamique des conteneurs à l'exécution. Concernant le déploiement de la plate-forme elle-même, nous supposons que la partie cliente de l'infrastructure est déjà installée sur les terminaux des utilisateurs. Actuellement, cette installation est faite à la main sur chacun des terminaux. Une solution de partenariat avec un fabriquant de PDA doit être étudiée. A plus long terme, nous souhaitons expérimenter d'autres exemples de services ubiquitaires, dans l'optique de proposer une approche dirigée par les modèles pour construire ces services, indépendamment des plates-formes d'exécution.

7. REFERENCES

[1] AppearNetworks, *The Appear Provisioning Server* http://www.appearnetworks.com.

[2] Ayed, D., Taconet, C., Bernard, G., *Deployment and Reconfiguration of Component-based Applications in AMPROS*. Proactive computing workshop (PROW 2004) - Finland, 2004.

[3] Briclet, F., Contreras, C., Merle, P., *OpenCCM : une infrastructure à composants pour le déploiement d'applications à base de composants CORBA*, Actes *conférence DECOR'04*, France, 2004.

[4] Buffalo Tech, *WLI-CB-G54A User Manual*, 2002.

[5] Compaq, *WL 110 Wireless Card User Manual*, 2001.

[6] D-link, *DWL-AG660 User Manual*, 2004.

[7] Edwards, W. K. *Core Jini Introduction*. Prentice Hall PTR, 1999.

[8] Endres, C., Butz, A., *A Survey of Software Infrastructures and Frameworks for Ubiquitous Computing*, Mobile Informations Systems Journal, 1(1), Jan-Mar 2005.

[9] Leydekkers, P., *Multimedia Services in Open Distributed Telecommunications Environments*. PhD Thesis CTIT, 1997.

[10] Marvie, R., Merle, P., Geib, J.M., Vadet, M., *OpenCCM : une plate-forme ouverte pour composants CORBA*, Actes conférence CFSE'2, p. 1-12, Paris, France, 2001. http://openccm.objectweb.org.

[11] Object Management Group, *CORBA Components Specification, version 3.0*, OMG TC Document formal/2002-06-65, USA, 2002.

[12] Object Management Group, *Trading Object Service Specification, verion 1.0*. OMG TC Document formal/2000-06-27, USA, 2000.

[13] Occello, A ., Dery-Pinna, A.M., *Sûreté de fonctionnement d'applications nomades construites par assemblage de composants*, UbiMob'05, Grenoble, France, 2005.

[14] Weiser, M ., *The computer for the 21st century*, Scientific American, 3(265):94-014, 1991.