RENPAR'15 / CFSE'3 / SympAAA'2003
La Colle sur Loup, France, 15 au 17 octobre 2003

OSGiTV : une plate-forme de déploiement d'applications de télévision interactive basée sur OSGi.

Stéphane Chomat, Didier Donsez

Laboratoire LSR, Institut IMAG, Université Joseph Fourier
Bat. C, 220 rue de la Chimie, Domaine Universitaire
BP 53, 38041 Grenoble Cedex 9, France
{Stephane.Chomat, Didier.Donsez }@imag.fr

Résumé
La télévision interactive est un nouveau domaine pour les développeurs d'applications. Une des principales différences avec les applications pour PC se trouve au niveau du déploiement des applications au travers de réseaux à diffusion (satellite, câble, réseau hertzien terrestre) avec ou sans lien de retour vers l'opérateur. Ce papier présente le gain qu'apporte OSGi pour déployer de telles applications dynamiques sur des réseaux diffusants. Ce papier le démontre au moyen d'un prototype de plate-forme de télévision interactive dans le respect des standards de la iTV.

Mots-clés : Déploiement logiciel, OSGi, Télévision Interactive.

---

1. Introduction

La télévision interactive (iTV) est un nouveau domaine prometteur pour les développeurs d'applications[1][2][3]. Comme les postes de TV sont omniprésents et que leur usage s'adresse au plus grand nombre de consommateurs, les postes de télévision interactive seront probablement un des points d'entrée majeurs vers les services en ligne. Les terminaux numériques de iTV de nouvelle génération sont maintenant capables de visualiser des documents xHTML et d'éxecuter des présentations Flash ou des applications Java appelé Xlets.
L'infrastructure du réseau de télévision interactive utilise principalement des technologies de diffusion uni-directionnelles comme le câble, le satellite ou le réseau hertzien terrestre (Figure 1). Le terminal peut parfois être connecté par modem (RTC, ADSL) à un réseau public. Ce lien offre alors une voie de retour vers les serveurs de l'opérateur de iTV notamment pour des applications transactionnelles tels que l'achat en ligne, les jeux en réseau, les paris ...
Actuellement, l'usager iTV (appelé abonné) souscrit auprès d'un opérateur à des canaux payants dont le flux est brouillé pour en contrôler l'accès. Les fournisseurs de services (jeux d'argent, banque, commerçant, ...) peuvent atteindre ainsi plusieurs millions de terminaux via un ou plusieurs opérateurs iTV. Jusqu'à présent, chaque opérateur loue des terminaux à peu près homogènes à ses abonnés et les administre en déployant le logiciel système (*firmware*) au redémarrage du terminal via son réseau diffusant (*broadcast*). Cependant, l'administration du parc de terminaux reste néanmoins peu précise (en comparaison avec un réseau d'entreprise) puisque l'usager ne raccorde pas toujours le modem de son terminal au réseau téléphonique pour assurer la voie de retour.
Dans un tel contexte, une application (i.e., Java Xlet) doit être déployée sur des millions de terminaux avec le minimum d'intervention du fournisseur du service associé à cette Xlet, de l'opérateur transportant son code et ses données et de l'usager final. L'intergiciel de télévision interactive (qui est pré installé sur le terminal avant la commercialisation) charge et installe l'ensemble des constituants de l'application (classes, composants, modules, ... de code et ressources) depuis le lien diffusant. L'installation peut être initiée par l'opérateur (c'est le cas de l'Xlet " Grille de Programme ") ou par l'usager quand

il sélectionne l'application depuis la grille de programme. Comme les ressources mémoire du terminal sont limitées, les constituants inutilisés trop longtemps doivent être supprimés sans interruption de service du terminal et des applications en cours de fonctionnement.
La télévision numérique terrestre (TNT) propose beaucoup de canaux et services gratuits, ce qui bouleverse le modèle économique de la iTV basé jusqu'à présent sur l'abonnement payant. Une des principales conséquences sera l'introduction sur le marché d'une grande variété de terminaux (du terminal bas de gamme aux consoles de jeux vidéo) achetés par l'usager et qu'il " administrera " lui-même.
Cette hétérogénéité nouvelle introduit de nouveaux défis pour les opérateurs et les développeurs d'applications de télévision interactive qui doivent désormais tenir compte de la grande palette de terminaux accessibles avec des capacités matérielles et des configurations logicielles très hétérogènes. Hors, les intergiciels de télévision interactive actuels (OpenTV,Canal+ MediaHighWay) pré-supposent qu'un opérateur déploie un parc de terminaux à peu près homogènes chez ses abonnés ; ils fixent l'environnement d'exécution logiciel de ces derniers et ne considèrent pas l'évolution de applications déployées. De plus avec ces intergiciels, la conception des applications de iTV reste très monolithique ce qui ne facilite pas l'adaptation de l'application à un environnement logiciel variable d'un terminal à un autre.

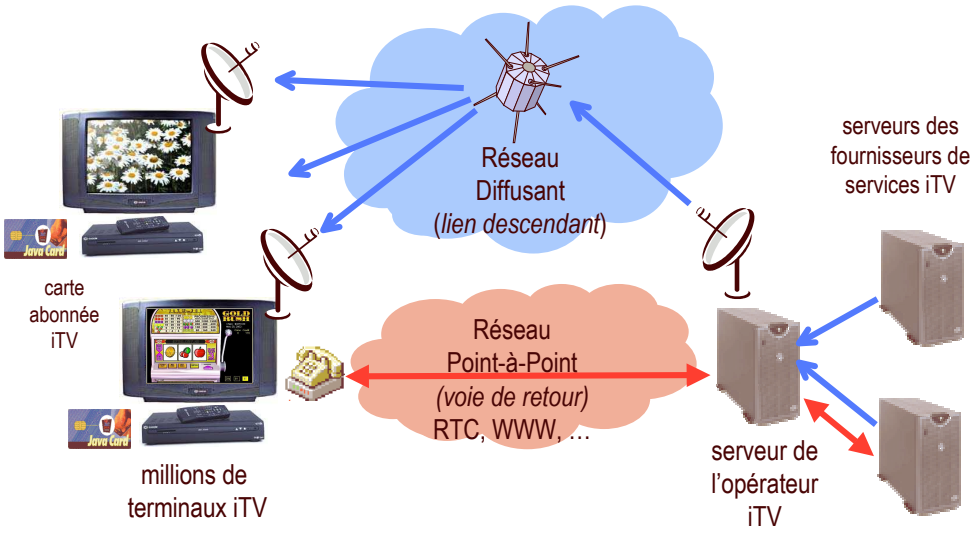

FIG. 1 – *infrastructure d'un réseau de télévision interactive*

Nous pensons qu'afin de déployer (c-à-d installer, démarrer, mettre à jour, retirer) correctement les applications de iTV, mais également les éléments de l'environnement d'exécution et du système d'exploitation, les terminaux de iTV requièrent une plate-forme de déploiement flexible d'applications dynamiques. Dans cet article, nous proposons une telle plate-forme qui étend OSGi pour le déploiement à très grande échelle et l'exécution automatique d'applications iTV dans le respect des principaux standards de ce domaine. Nous avons implémenté et validé cette plate-forme par un démonstrateur dans un environnement de simulation d'une infrastructure réseau de iTV.
La suite de l'article est organisée de la manière suivante : nous présenterons premièrement le contexte technologique dans lequel se place le travail présenté avec un survol des principaux standards qui régissent les intergiciels iTV et de la plate-forme OSGi qui sert de base à notre plate-forme de déploiement. La section 3 présente notre modèle d'applications dynamiques. La section 4 décrit ensuite l'architecture de la plate-forme de déploiement d'applications sur les terminaux de iTV et détaille son fonctionnement. La section 5 discute de l'implémentation de cette plate-forme par un prototype et de l'environnement de simulation. La section 6 positionne ce travail par rapport aux travaux relatifs au déploiement d'applications. Enfin, nous concluons avec des perspectives à ce travail.

2. Contexte technologique

L'industrie de la télévision numérique interactive est régie par plusieurs standards complémentaires, parfois concurrents, voir redondants et dont les documents de spécification comprennent souvent plusieurs milliers de pages. Nous ne parlerons dans la première section que de ceux sur lesquels ce travail s'appuie et que nous tentons de respecter dans la conception de la plate-forme. La seconde sous-section introduit la spécification OSGi sur laquelle s'appuie notre plate-forme pour la livraison et l'exécution des services fournis par les composants des applications à déployer.

2.1. Standard de la télévision interactive

Un des socles fondateurs de la télévision interactive est le standard MPEG2-TS (Transport Stream) [4] qui normalise le transport (multiplexé) des flux audio/vidéo sur des réseaux diffusants (câble, satellite, terrestre). Ces flux sont décrits au moyen de plusieurs tables également diffusées dans les flux. Certaines tables sont standardisées comme la Table d'Information des Programmes (PIT) et d'autres, dites privées, sont utilisées par l'opérateur, par l'intergiciel ou par les applications. MPEG2-TS spécifie également le transport de fichiers de données (document HTML, classe Java, icône, ...) notamment grâce à un système de fichiers pour réseaux diffusants [5]. Le serveur de système de fichiers appelé Carrousel réémet cycliquement les fichiers. De leurs cotés, les (millions de) clients peuvent lire les fichiers diffusés mais la lecture d'un segment d'un fichier est bloquante jusqu'à leur prochain ré-émission. Chaque client peut gérer un cache local des fichiers recus.

DVB (Digital Video Broadcast), un consortium européen d'industriels de la iTV, s'est consacrée ces dernières années à la standardisation d'un environnement d'exécution des applications interactives appelé MHP (Multimedia Home Platform)[6]. Pour cela, DVB complète MPEG-2 des tables de signalisation comme l'Application Information Table (AIT) dont nous nous servirons pour déployer les applications dans notre plate-forme. DVB-J définit l'environnement d'exécution pour des applications iTV développées en Java. DVB-J s'appuie entre autres sur les paquetage d'HAVi (Home Audio-Video Interoperability) pour la construction d'interfaces utilisateurs adaptées aux terminaux audio-vidéo de divertissement (*entertainment*).

JavaTV[7] de Sun, est largement inspiré de DVB-J et spécifié un environnement d'exécution d'applications iTV écrites en Java. JavaTV définit notamment le cycle de vie de ces applications appelées Xlet. DVB-MHP et JavaTV s'appuient sur l'environnement de Personal Java basé sur le JDK1.1. Cependant, Sun compte remplacer Personal Java par deux profiles récents, Personal Basic et Personal, de J2ME/CDC [8].

2.2. Open Services Gateway Initiative (OSGi)

Open Services Gateway Initiative (OSGi) est une spécification ouverte pour la définition d'une plate-forme de déploiement, de gestion et d'exécution des services administrés à distance dans des environnements embarqués tels que des passerelles résidentielles, industrielles ou véhiculaires. La spécification OSGi [9][10] définit une passerelle de services (*service gateway*) qui permet à une diversité de services logiciels d'être chargés, exécutés, mis à jour sur des ordinateurs enfouis tels que des PCs industriels, passerelles résidentielles dédiées, distributeurs automatiques, ordinateurs de bord de véhicule, ... Ces services logiciels peuvent être des pilotes de périphériques matériels reliés à la machine par un réseau domotique d'une habitation, des bus de terrains dans un véhicule ou une fabrique, et peuvent exécuter des fonctions de haut niveau (réguler la climatisation d'une habitation, " réagir " à une intrusion dans une pièce, ...)

Les trois principaux concepts de OSGi sont la passerelle de services, les bundles, les services. Les bundles sont des unités de livraison du code à installer versionné (i.e. des packages Java et bibliothèques natives) et de services à activer. Un service actif (e.g. un pilote de périphérique) publie une ou plusieurs interfaces. Il peut aussi requérir d'autres services actifs via leurs interfaces. La passerelle des services offre un environnement de déploiement des bundles et d'exécution des services. La passerelle n'autorise l'activation des services d'un bundle installé que si toutes les dépendances de packages sont résolues. Les contraintes de compatibilité ascendante des versions de packages sont également vérifiées. Les services sont enregistrés avec un ensemble de propriétés auprès du registre de passerelle. La recherche de services se fait par courtage en filtrant ceux-ci au moyen de leurs propriétés.

OSGi ne définit pas la notion d'application. Cependant, une application peut être vue comme un graphe

de dépendances de services avec un service racine démarrant l'exécution d'une *thread*. Dans le cas d'un serveur Web embarqué, chaque racine est un service implémentant l'interface *javax.servlet.http.HttpServlet*. Nous verrons que dans notre plate-forme, chaque racine d'application est un service implémentant l'interface de cycle de vie des *javax.tv.xlet.Xlet*.
Pour déployer un service, l'administrateur de la passerelle OSGi charge à partir du système de fichiers local ou d'un serveur distant le ou les bundles contenant les codes de classes requis par le service. Les bundles peuvent être ensuite installés, démarrés, arrêtés, mis à jour, redémarrés et finalement retirés sans que la plate-forme d'accueil ne soit arrêtée (i.e. *reboot*). Le démarrage d'un bundle correspond généralement au lancement des services attachés au bundle.
OSGi cible principalement le développement d'applications dont l'architecture peut évoluer dynamiquement au cours de son exécution. En effet, de nouveaux services peuvent apparaître ou disparaître correspondant à l'ajout ou au retrait des périphériques ou le démarrage et l'arrêt de services de haut niveau. De fait ; la programmation de classes du bundle impose à son développeur de coder des *listeners* à l'affût du chargement d'états (arrêt/démarrage) des bundles et services dont dépend ce bundle.

3. Modèle d'applications iTV dynamiques

Les intergiciels actuels de télévision interactive ne permettent que le développement d'applications dont la structure monolithique est figée au moment du développement. L'application est un ensemble fermé de classes. Le développeur fait l'hypothèse que son application s'exécutera sur un environnement d'exécution adapté aux terminaux déployés par l'opérateur. Bien que l'application possède un numéro de version, ses classes sont dépourvues d'informations de version. Le déploiement d'une application consiste à diffuser ses classes et ses ressources par le système de fichier carrousel et à ajouter une entrée dans l' Application Information Table (AIT) qui est aussi diffusée.
Ce modèle d'applications que nous appelons statique pose plusieurs problèmes au cours du cycle de vie de l'application. Premièrement, tout changement dans le code de l'application (i.e. ajout/retrait de fonctionnalité, mise à jour, ...) oblige l'opérateur à redéployer l'ensemble des classes de l'application. Ce redéploiement ne peut être pris en compte coté terminal que lorsque l'application est arrêtée. Il est nécessaire de continuer à diffuser les classes de plusieurs versions d'applications. Deuxièmement, l'application est développée pour un environnement d'exécution donné. Or dans le contexte de l'ouverture des terminaux à de multiples opérateurs, il est difficile de figer une application pour un environnement d'exécution donné.
Le modèle d'applications dynamiques que nous proposons, structure l'application sous la forme d'un graphe dynamique de services [12]. Les services peuvent être ajoutés, mis à jour ou retirés de la structure de l'application au cours de son exécution tant que les règles de cohérence de l'application sont respectées. L'exemple de la figure 3 illustre l'évolution dynamique de l'architecture d'une application de iTV. Cette application permet à l'usager de miser de l'argent à des jeux de casino (bandit manchot, poker, black jack, ...) en restant chez lui. Le fournisseur de ce service de T-Casino remet à chaque usager joueur une carte à puce sécurisant l'argent des paris et des gains et un moteur de tirage aléatoire. Le rechargement de la carte (interface *Refill*) en argent pour les paris et la récupération des gains (interface *CashOut*) se font, de manière occasionnelle, avec le serveur du fournisseur T-Casino par la voie de retour quand celle-çi existe. L'invocation des méthodes sur le serveur distant passe par un talon RMI. Selon le même principe, l'appel des méthodes de la carte T-Casino passe par un talon JavaCard-RMI. Un moteur de tirage aléatoire non sécurisé est prévu pour permettre l'utilisation de ce service en mode démonstration. Dans notre modèle dynamique, l'application est conçue comme un ensemble de services (Fig 3). L'application T-Casino est cohérente à partir du moment où le service racine *TCasinoXlet* peut utiliser au moins un service de moteur de tirage (interface *Random*) et au moins un service de moteur de jeux (interface *GameEngine*). Au cours de l'exécution, l'insertion de la carte T-Casino provoque le déploiement et l'activation du mandataire carte (par exemple un talon JavaCard-RMI) qui publie un service supplémentaire *Random* : l'application doit désormais tenir compte de ce nouveau service pour réaliser les paris avec la carte. L'opérateur peut également déployer de nouveaux moteurs de jeux (roulette, pile ou face ...), l'application doit modifier la liste des nouveaux jeux disponibles.
Les constituants des intergiciels classiques de iTV sont figés jusqu'à la prochaine mise à jour du *firmware* qui oblige un redémarrage du terminal. Cette condition peut être pénalisante quand l'usager bascule (c-

à-d *zappe*) d'une application d'un opérateur à une autre d'un opérateur concurrent puisque des éléments de l'environnement d'exécution pour changer d'un opérateur à l'autre (ex. les classes d'accès au module de débrouillage du flux). L'intergiciel de iTV peut profiter du modèle dynamique pour déployer ses éléments dynamiquement (bibliothèques de l'environnement d'exécution, pilote des périphériques) dynamiquement comme les constituants des applications.

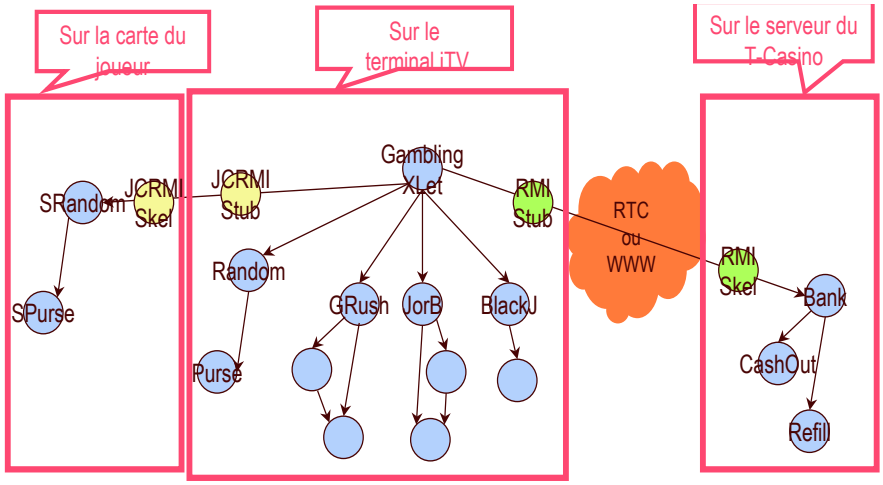

FIG. 2 – *exemple d'application iTV de TCasino*

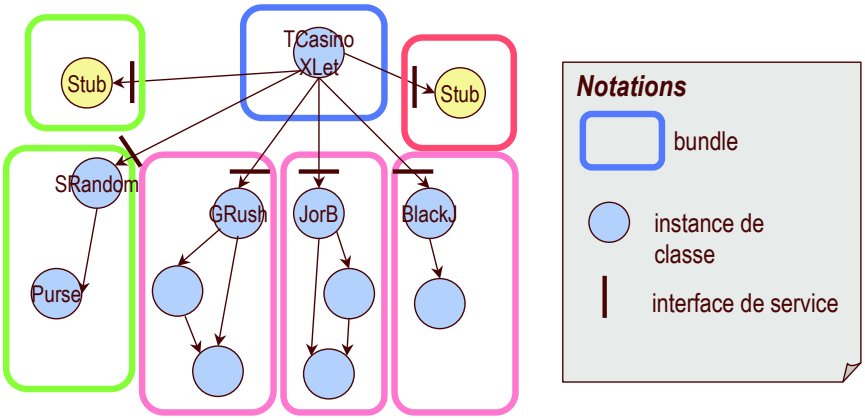

FIG. 3 – *conception orienté service et conditionnement de l' application terminal TCasino*

4. Notre plate-forme de déploiement

OSGi est un bon point de départ pour construire et déployer des applications dynamiques. Cependant, OSGi possède plusieurs inconvénients. De manière générale, la programmation de l'enregistrement des services et de la prise en compte de leurs retrait restent des tâches fastidieuses pour le développeur. De plus, il n'est pas possible d'exprimer la structure d'une application en exprimant les dépendances statiques ou dynamiques entre les services qui composent celle-ci.

Notre plate-forme appelé OSGiTV se base sur ServiceBinder [11], un travail mené au sein de notre laboratoire, qui propose un modèle à composant au dessus d'OSGi. ServiceBinder reprend les principaux concepts des modèles à composants comme CCM. Il définit les notions de fabriques et des instances qui fournissent et requièrent des services caractérisés par leurs interfaces. Pour ServiceBinder, les services standards d'OSGi et les services patrimoniaux sont alors vus comme des composants singletons. Chaque composant est conditionné dans un bundle OSGi standard additionné d'un fichier de méta-données. A partir des descriptions fournies dans ce fichier de méta-données, ServiceBinder gère le cycle de vie des instances et automatise les liaisons (*binding*) entre celles-ci ainsi que leur rupture (*unbinding*) en cas de retrait d'un autre composant. La liaison peut être décrite statique ou dynamique. Une liaison statique est fixée lors de la création de l'instance. Par contre, la liaison dynamique peut être rompue, replacée, complétée tout au long de l'existence d'une instance. Les liaisons sont contraintes par des cardinalités qui définissent des règles de cohérence des instances dans l'application [12].
Une autre inconvénient majeur d'OSGi concerne le démarrage des bundles. D'un part, un bundle installé ne peut démarrer et activer ses services que si les packages qu'il requiert sont déjà installés. D'autre part, dans le contexte de la télévision interactive, quand un service requiert (avec ou sans ServiceBinder) l'usage d'un autre service, il est fort probable que le bundle qui conditionne le service en question ne soit pas installé et démarré car il y a potentiellement plusieurs centaines de bundles rendus disponibles par le ou les opérateurs. Or ces bundles ne peuvent pas être tous simultanément installés et démarrés sur un terminal contraint par des ressources mémoire limitées.
Pour cela, notre plate-forme OSGiTV met en œuvre une opération de chargement et de démarrage à la demande des bundles/composants et de leurs services. Cette opération est transparente au service demandeur qui se fait intercepter ces demandes de services par le service BundleOnDemand. BundleOnDemand installe et démarre les bundles proposant les services demandés. Cette opération est récursive quand les services des bundles demandent à leur tour des services non installés.
Dans la suite de cette section, nous décrirons les différents éléments de la plate-forme permettant le déploiement et l'activation des applications et de leurs composants. Nous commencerons par lister les tables nécessaires au déploiement puis par décrire les outils coté opérateur et enfin le fonctionnement des services du coté du terminal.

4.1. Tables d'information

Cette plate-forme respecte et étend la signalisation définie dans DVB-MHP [6]. Les applications sont identifiées au moyen de la table standard (ie publique) AIT. Chaque entrée correspond au nom d'une application et à une action à réaliser lors de la réception de la table. Ces actions sont l'ajout de l'application à la liste des applications accessibles à l'usager depuis la " grille des programmes " (*store*), son retrait (*unstore*) ou son activation automatique (*autostart*).
La signalisation standard est complétée par plusieurs tables privées pour les besoins du déploiement.

- La table d'information des bundles (BIT : Bundle Information Table) liste les informations décrivant les bundles que l'opérateur diffuse. Chaque entrée contient la liste des packages versionnés qui sont importés et qui sont exportés par le bundle, la liste des services fournis avec leurs propriétés de courtage et la liste des services requis avec leur filtre de courtage. Les deux listes de package sont extraites du fichier de manifeste du bundle tandis les deux listes de services sont extraites de fichier de méta-données de ServiceBinder quand celui-ci est utilisé pour gérer les services du bundle. Chaque entrée contient une ou plusieurs URL pour le chargement du bundle pour permettre des sources de chargement alternatives au réseau diffusant (par exemple par la voie de retour ou en pair à pair avec d'autres appareils dans le logement).
- La table d'information de pilotes (DIT : Driver Information Table) liste les informations relatives aux pilotes de périphériques. Chaque entrée contient les informations nécessaires au mécanisme de raffinement de pilotes (ie DAM : Device Access Manager) conformément à la spécification OSGi. Cette table est utilisée par le localisateur de pilote du DAM du terminal.

Ces différentes tables peuvent être diffusées par l'opérateur, pré-stockées sur le terminal lors de sa commercialisation, téléchargées par le modem de la voie de retour, ou bien encore chargées depuis une carte à puce lors de son insertion dans le terminal. Les contraintes de taille des paquets du réseau rendent préférable la fragmentation de ces tables en plusieurs morceaux de une ou plusieurs entrées.

4.2. Coté opérateur

Avant de diffuser les bundles des applications et de l'intergiciel, l'opérateur doit réaliser deux opérations sur ceux-ci. Premièrement, l'opérateur extrait les informations afin de construire les deux tables privées (BIT,DIT) et fragmente les 3 tables (AIT, BIT, DIT) en plusieurs paquets. Deuxièmement, une classe d'activation générique d'interception est insérée dans le fichier Jar de chaque bundle afin de permettre au service BundleOnDemand d'installer des bundles publiant les services requis lorsque ceux-ci sont demandés.

Les fragments de ces tables et les fichiers Jar des bundles sont alors déposés sur le serveur (*CarrouselBroadcaster*) du système de fichier carrousel de l'opérateur qui les diffuse cycliquement vers les terminaux.

4.3. Coté terminal

La plate-forme coté terminal tourne au dessus d'un *framework* OSGi standard sur lequel sont préinstallés les bundles suivants. La figure 4.3 schématise ces principaux éléments.

Le récepteur du système de fichier carrousel (*CarrouselReceiver*) reçoit et ré-assemble les segments des fichiers diffusées. Ces fichiers sont alors cachés dans le cache du terminal. Ce cache, selon les capacités du terminal, peut être en mémoire primaire ou en mémoire secondaire. La mémoire secondaire peut être une mémoire flash de faible capacité comme un disque dur de quelques dizaines de giga-octets pour la dernière génération de terminaux comme la PiloTime de Canal Satellite. Les fichiers ne sont pas évincés du cache quand ils sont en cours d'utilisation par d'autres éléments de la plate-forme.

Pour leurs parts, les tables d'information sont passées à des TableLocators (un par type de table) qui génèrent un objet d'événement pour chaque entrée d'une table. Cet objet est alors passé au EventDispatcher qui notifie les abonnés de ce type d'événement.

Le gestionnaire d'applications, XletManager, s'abonne aux événements de type AI et met à jour sa base d'information. XletManager démarre une application quand il reçoit un objet AI positionné à *autostart.* Pour cela, il délégue l'installation de l'application au service d'installation de BundleOnDemand en recherchant un service *javax.tv.xlet.Xlet* qualifié par propriété *XletName* dont la valeur est l'identifiant utilisé dans l'AIT.

Le gestionnaire d'installation à la demande, BundleOnDemand, permet d'installer et de démarrer récursivement le bundle racine de la Xlet et ses dépendances de packages. Pour cela, il utilise sa base d'information qui est alimentée par les événements BI (Bundle Information) notifiés par l'EventDispatcher. L'installation des dépendances peut être récursive et il peut exister plusieurs choix d'installation quand deux ou plusieurs bundles exportent le même package. La stratégie de sélection privilégie le bundle dont le numéro de version de package est le plus élevé en faisant l'hypothèse de la compatibilité ascendante. Si plusieurs bundles restent encore en lice, la stratégie prend le bundle qui fournit le plus grand nombre de services qualifiés vérifiant les filtres de services requis par le bundle à installer. Cette stratégie tente de minimiser le nombre de bundles installés sur le terminal. Cette stratégie par défaut peut être changée par une autre stratégie de sélection. BundleOnDemand fournit également un service que les classes d'interposition insérées dans les bundles interrogent pour récupérer les références de service requis. Ce service provoque l'installation de bundles listés dans la BIT quand ceux-ci proposent le service qualifié demandé.

Le gestionnaire des pilotes de périphériques, Device Access Manager, respecte la spécification OSGi pour l'installation des pilotes de périphériques branchés sur le terminal. Cependant dans le contexte d'un réseau diffusant, notre plate-forme le complète par un localisateur de pilote (DriverLocator) en utilisant sa base d'information locale sur les pilotes disponibles. Cette base est alimentée par les objets DI notifiés par l'EventDispatcher. Conformément à la spécification, le DAM met en œuvre un mécanisme dit de raffinement pour tester l'installation de pilotes de bas niveau puis passer à l'installation de pilotes de plus niveau.

4.4. Démarrage d'une application

Le démarrage d'une application peut être initié par l'opérateur, par l'usager à l'aide de la grille de programme ou par l'insertion d'une carte à puce. Dans ces 3 cas, c'est le XletManager qui se charge du démarrage à la réception d'un objet AI marqué *autostart.*

Dans le premier cas, l'opérateur positionne à *autostart* l'entrée de application dans l'AIT diffusé C'est

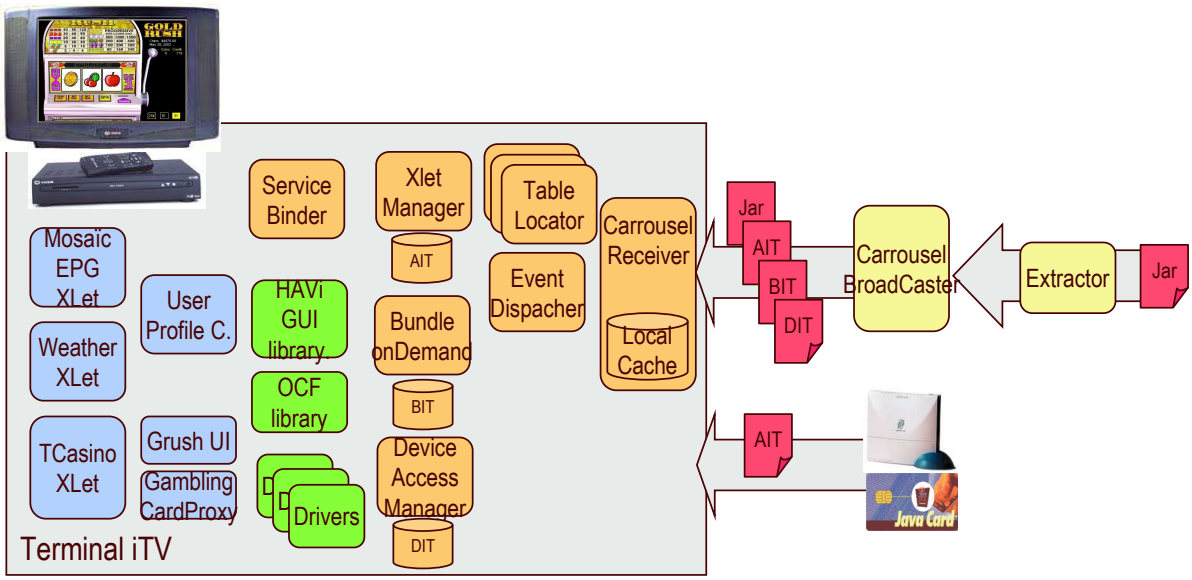

FIG. 4 – *architecture de la plate-forme de déploiement OSGiTV*

d'ailleurs le cas de l'application (Xlet) " Grille de Programme " qui accueille l'usager quand il démarre sur terminal.
Dans le deuxième cas, l'usager utilise l'IHM de la Xlet " Grille de Programmes " pour lancer une autre Xlet. La Xlet " Grille de Programmes " génère alors un événement AI marqué autostart qui est alors notifié au XletManager.
Le démarrage d'une application peut être aussi initié par l'insertion d'une carte à puce dans le lecteur du terminal. C'est d'ailleurs le cas de l'application de T-Casino illustrée par les figures 3 et 3. Le démarrage intervient à la suite du processus de raffinement du périphérique carte à puce qui charge et démarre un mandataire offrant un service d'accès au moteur de pari aléatoire mais également un service CardAITLocator qui génère un événement AI pour le lancement immédiat (*autostart*) de la Xlet TCasino. Le retrait de la carte provoque la désinstallation des bundles associés à cette carte et l'arrêt de leurs services. L'arrêt du CardAITLocator génère un objet événement demandant le retrait (*unstore*) de la Xlet de TCasino de la base des applications disponibles.
Les éléments de l'environnement d'exécution sont donc chargées soit sous la forme de pilotes de périphériques (cas d'OpenCard Framework pour les lecteurs de cartes) soit comme des bundles patrimoniaux (HAVi pour l'interface utilisateur). Ces bundles sont utilisés et partagés par plusieurs applications.

5. Réalisation et Expérimentations

Notre plate-forme de déploiement OSGiTV a été réalisée en Java au dessus d'une passerelle respectant la spécification 2.0 d'OSGi. Nos tests ont été effectués sur OSCAR[13], une plate-forme open-source maintenue au laboratoire, pOSCAR un portage d'OSCAR sur le JRE 1.1 que nous avons réalisé spécialement pour ce projet et sur JES (Java Embedded Server). La machine utilisée est un PC Pentium II de bureau tournant sous Linux ou Windows NT/2000, dont les performances avoisines celles du PiloTime, le nouveau terminal numérique de Canal Satellite.
Les éléments de l'environnement d'exécution comme les packages graphiques d'HAVi et les packages d'OCF ont été " emballés " dans des bundles OSGi exportant les packages nécessaires aux applications. Ceux-ci sont donc déployés comme tout autre bundle.
L'usage d'un réseau satellitaire étant difficile dans le cadre d'une expérimentation académique, nous avons implémenté les parties diffuseur et récepteur du système de fichier carrousel sur IP Multicast afin de simuler la diffusion. Le diffuseur segmente les fichiers et les envoie dans des datagrammes IP

Multicast. Le récepteur reçoit et ré-assemble les segments des fichiers qui sont conservés dans un cache local. Le récepteur permet l'abonnement à des événements sur l'arrivée de segments de fichiers. Les fichiers peuvent être lu au moyen du schéma d'URL de la forme *mcast://classDaddress:port/path* qui est ajouté à la plate-forme *java.net.URLConnection* de JRE.
Nous avons testé le déploiement simultané d'une quinzaine d'applications Xlet simple de démonstration dont une application " grille des programmes " et l'application de T-Casino qui peut être déployée et activée par l'insertion d'une carte à puce contenant une " cardlet " de stockage de l'AIT et la " cardlet " de moteur de tirage aléatoire.
La génération des tables privées à partir des bundles est actuellement réalisée " manuellement " mais elle sera automatisée sous la forme d'une tache ANT.
Il est difficile d'obtenir des informations et des mesures sur les principaux intergiciels de iTV. La comparaison de leurs performances et leur empreinte mémoire n'a pas pu être effectuée avec notre plate-forme académique. Cependant, nous donnons à titre indicatif suivant les tailles des archives Jar (non compressé) des bundles des éléments de notre plate-forme données dans la table 1. Les éléments suivis d'une astérisque correspondent à des développements originaux. Les autres sont des packages extérieurs qui ont parfois été légèrement modifiés pour être emballés dans des bundles OSGi.

Passerelle	
API OSGi	42
Passerelle OSGi OSCAR	276
Serveur	
CarrouselBroadcast multicast *	68
Noyau de la plate-forme	
CarrouselReceiver multicast	52
EventDispatcher *	18
BundleOnDemand *	43
ServiceBinder	31
XletManager *	76
TableLocator(s) *	34
Périphériques	
DeviceAccessManager+McastDriverLocator *	34
GemplusSerialDriver (lecteur de carte)	390
OCFDriver	392
Environnement pour les applications	
JavaTV	294
DisplayManager (version allégée de HAVi)	200
Exemples d'applications de démonstration	
MosaïcEPGXlet *	22
TCasinoXlet+GoldRush *	406

TAB. 1 – *taille des éléments constituant la plate-forme (en Ko)*

6. Travaux relatifs

Ce travail est à l'intersection des recherches sur les composants logiciels[14], sur la conception orienté service[15], sur les applications dynamiques[12] et sur le déploiement de logiciel à grand échelle. Cependant, nous avons choisi de positionner notre travail par rapport à ce dernier domaine.
Différentes recherches ont été menées dans le domaine du déploiement de logiciel. Carzaniga et al. [16] présente le domaine de déploiement logiciel et liste les principales techniques. Plusieurs solutions existent dans l'industrie et dans le monde académique. Software Dock et Tivoli sont des solutions de

déploiement de logiciels sur les sites d'un entreprise. Software Dock est basé sur les agents mobiles mais selon un modèle client/serveur. Tivoli peut utiliser le modèle push pour le déploiement des ressources informatiques vers un nombre fini de sites. Ces 2 solutions sont plutôt adaptées au contexte bien maîtrisé d'un réseau d'entreprise.
Java Web Start [17] propose une solution de déploiement d'applications Java sur des stations de travail/PC. Il permet de déployer les applications Java et leurs bibliothèques mais également les environnements d'exécution (JRE) requis. Java Web Start définit seulement des dépendances statiques des packages ce qui est inadapté aux applications dynamiques que nous souhaitons déployer.
On peut souligner que les plate formes à composants d'entreprise (EJB [18], CCM [19], .NET [20]) traitent plus ou moins le volet déploiement dans leurs modèles. Cependant, ces modèles sont principalement focalisés aux applications tournant sur les serveurs d'entreprise bien qu'il existe des versions de .NET et CCM s'exécutant sur des assistants personnels.
OSGiTV reprend des idées du projet RNRT CESURE [21][22] pour le déploiement d'applications à partir d'une carte à puce. Cependant dans CESURE, le tisseur de composants CCM est embarqué dans la carte pour des questions de sécurité et de personnalisation ; dans OSGiTV, le " tisseur " qui est ServiceBinder combiné à BundleOnDemand, doit fonctionner en l'absence de carte.

7. Conclusion et perspectives

Dans cet article nous avons présenté OSGiTV une plate-forme de déploiement d'applications dynamiques de télévision interactive. Cette plate-forme se base sur la spécification OSGi pour le conditionnement et la livraison des composants logiciels des applications à déployer et de leur environnement d'exécution. Le modèle de composants d'OSGiTV reprend celui de ServiceBinder qui automatise la liaison et le retrait des services qui composent les applications dynamiques. Une des principales contributions de ce travail est l'automatisation des installations de bundles pour OSGi à partir des dépendances de packages et des demandes de services. Cette fonctionnalité est indispensable dans le contexte de terminaux de iTV dépourvus d'administrateur.
Les perspectives de nos travaux sont multiples. En premier, la plate-forme courante doit être complétée par des fonctions de personnalisation et de sécurité. En effet, l'opérateur doit pouvoir limiter l'usage de certaines applications à une part de ses clients (abonnés ou non) en fonction de son profil ou de ses droits. En second, le retrait des services et des bundles est actuellement limité aux seules Xlet et aux bundles chargés par le Device Access Manager.
Ce travail nous a également conduit à nous intéresser à un modèle à composants au dessus de OSGi permettant l'ajout d'aspects non fonctionnels (sécurité, persistance, comptage des usages, ...) aux composants déployés. Les services non fonctionnels correspondants sont eux même des composants conditionnés et déployés dynamiquement dans des bundles quand ceux-ci sont requis par un service (fonctionnel ou non) de la plate-forme. Pour cela, nous nous appuyons sur les travaux autour des conteneurs extensibles [23] et des tisseurs d'intergiciels Fractal [24] et Avalon[25].

Bibliographie

1. Press L. – The Internet and interactive television – Communications of the ACM, Volume 36 , Issue 12, pp 19 – 23, Décembre 1993
2. O'Driscoll G – The essential guide to Digital Set-Top Boxes and Interactive TV – Ed. Prentice Hall, 2000
3. O'Driscoll G. – The essential guide to Home Networking technologies – Ed. Prentice Hall, 2001.
4. Sarginson. P.A. – MPEG-2, Overview of the system layer – BBC R&D report, 1996.
5. Balabanian V., Casey L., Greene N., Adams C.. – An introduction to digital storage media-command and control – IEEE Communications Magazine, Volume: 34, Issue: 11, novembre 1996, . pp 122-127
6. Digital Video Broadcast (DVB) – Multimedia Home Platform 1.0.2, DVB BlueBook A057 Rev.2 – European Broadcasting Union, février 2002.
7. Calder B., Foote B., Kyrnitzke L., Rivas D., Van Loo J., Ye. T. – Java TV API technical overview, v1.0 – Sun MicroSystems, juillet 2000.
8. Sun – JSR 129, Personal Basic Profile – http://www.jcp.org/jsr/detail/129.jsp
9. Open Service Gateway initiative (OSGi) – Open Service Gateway Specification, version 3.0 – mars 2003, http://www.osgi.org

10. Chen, K., Gong, L. – Programming Open Service Gateways with Java Embedded Server Technology – Ed. Addison Wesley, Août 2001, ISBN 0201711028.
11. Cervantes, H., Hall, R. S. – ServiceBinder – http://gravity.sourceforge.net/servicebinder/
12. Cervantes H, Hall R.S. – Automating Service Dependency Management in a Service-Oriented Component Model – Actes du ICSE CBSE6 Workshop Portland, Oregon, USA, May 3-4, 2003
13. Hall, R. – OSCAR, Open Service Container Architecture – http://oscar-osgi.sourceforge.net/
14. Szyperski, C. – Component Software: Beyond Object-Oriented Programming – Ed. Addison-Welsey, Décembre. 1997, ISBN 0-201-17888-5, 1997
15. Bieber G. , Carpenter J. – Introduction to Service-Oriented Programming – whitepaper, septembre 2001, http://www.openwings.org/download/specs/ServiceOrientedIntroduction.pdf
16. Carzaniga A., Fuggetta A., Hall R.S., van der Hoek A., Heimbigner D., A.L. Wolf. – A Characterization Framework for Software Deployment Technologies – Technical Report CU-CS-857-98, Dept. of Computer Science, University of Colorado, April 1998.
17. Marinilli M. – Java Deployment with JNLP and WebStart – Ed SAMS, Novembre 2001, ISBN 0-672-321823
18. SUM MicroSystems – Enterprise JavaBeans 2.1 Specification – http://java.sun.com/products/ejb/
19. MicroSoft – .NET Framework – http://msdn.microsoft.com/netframework/
20. Object Management Group – CORBA Component Model Specification V3.0 – http://www.omg.org, Juin 2002
21. Pellegrini MC, Potoniée O., Marvie R., Jean S., Riveill M. – CESURE : une plate-forme d'applications adaptables et sécurisées pour usagers mobiles – dans Evolution des plate-formes orientées objets répartis, issue spécial de Calculateurs Parallèles, Ed. Hermès, Paris, France, September 2000.
22. Potonniée O – Ubiquitous service access for nomadic users – Présentation à la Conférence JavaOne 2002, Mars 2002, San Francisco, USA
23. Duclos F., Estublier J., Morat P. – Environnement pour la gestion d'aspects non-fonctionnels dans un modèle à composants – Actes de la conférence ICSSEA, Paris, France. 4-6 Décembre 2001.
24. ObjectWeb.org – The Fractal Framework – http://fractal.objectweb.org
25. Apache.org – The Avalon Framework – http://jakarta.apache.org/avalon