Courtage et déploiement dynamiques de composants pour l'infrastructure d'équipements UPnP

Didier Donsez
Laboratoire LSR, Equipe ADELE
Université Joseph Fourier
BP 53, F38041 Grenoble Cedex 9,
didier.donsez@imag.fr

RÉSUMÉ
La banalisation des réseaux domestiques et la généralisation des équipements communicants font penser que la domotique devient peut-être enfin une réalité pour un marché de masse. Le contrôle des équipements passe par plusieurs types de points de contrôle spécialisés ou génériques. Cet article s'intéresse au courtage et au déploiement dynamique des composants logiciels destinés aux points de contrôle dans le contexte de réseaux d'équipements UPnP et aux passerelles entre UPnP et les micro-mondes. Un prototype réalisé en Java sur OSGi valide les concepts.

Mots-clés
UPnP, Points de contrôle, Courtage, Déploiement, Composant.

ABSTRACT
The generalization of communicating home equipments fosters the home automation becomes definitively a mass market. Equipment control requires several specialized controls or generic ones. This article is interested in broking and the dynamic deployment of software components implementing UPnP control points. It also addresses the bridge between UPnP and the world of micro-equipments. A prototype based on OSGi validates our proposition.

Categories and Subject Descriptors
D.2.13 **[Reusable Software]**: *Reusable libraries*. C.2.4 **[Distributed Systems]***: Distributed applications.*

General Terms
Algorithms, Design, Experimentation.

Keywords
UPnP, Control point, Trading, Deployment, Component.

1. INTRODUCTION
La domotique devient peut-être enfin une réalité pour un marché de masse. Jusqu'à présent, le marché de la domotique était *balkanisé* entre une multitude de fabricants d'équipements. Ceci avait certainement freiné son essor. Une des principales raisons était qu'il était jusqu'alors très difficile à l'intégrateur (architecte, installateur…) de réaliser pour ses clients une solution complètement intégrée couvrant tous les types d'équipements (climatiseur, volet roulant, centrale d'alarme, détecteur de présence…). L'émergence de protocoles ouverts largement répandus outre-atlantique tel que X10 et la généralisation de connectivité domestique IP filaire et sans-fil permet de penser qu'une nouvelle ère arrive pour la domotique [12]. En effet, des technologies de découverte des équipements telles que UPnP, mDNS, EchoNet, permettent l'ajout et le retrait dynamiques d'équipements dans l'environnement domotique sans intervention (configuration, administration) de l'utilisateur final.

Universal Plug and Play (UPnP) Forum [11] est un consortium industriel ouvert qui s'est formé en 1999 pour la définition de standards simplifiant la mise en réseau d'équipements communicants dans les maisons et dans les entreprises (*SOHO : Small Office Home Office*). UPnP Forum a publié une première version des protocoles réseaux requis et un certain nombre de définitions standards d'équipements et de services associés. UPnP est une plateforme distribuée de services dynamiques pour des équipements (télévision, lecteur de DVD, volets déroulants…) et des points de contrôle (*PDA*, télévision…) connectés entre eux par un réseau adhoc IP filaire ou sans-fils. UPnP définit les protocoles réseau permettant de la détection (et le retrait) dynamique des équipements, l'utilisation des services qu'ils fournissent par les points de contrôle et la notification des changements de valeur des variables d'état associées aux services. Dans cette plateforme, certains équipements sont en réalité des passerelles entre des « micro-mondes » d'équipements dialoguant selon d'autres protocoles (X10, Konnex, IEEE 1345…) et des points de contrôle UPnP. Elles intègrent dans un réseau UPnP des équipements de faible coût ne pouvant pas embarquer une pile IP.

Tout équipement UPnP (*UPnP device*) s'auto-décrit selon le modèle hiérarchique. Cette description (qui est exprimée dans une grammaire XML) comporte la liste des services qu'il fournit et éventuellement la liste d'autres équipements qu'il embarque. Chaque service d'un équipement est lui-même typé. Il comporte des variables d'état et des actions. Une variable peut-être, par exemple, le niveau sonore d'un téléviseur et une action peut-être d'augmenter ce niveau, une autre de le diminuer et une dernière de passer ou de sortir du mode muet. Certaines variables peuvent notifier leur changement de valeur aux points de contrôle qui s'abonnent à ces notifications.

Les points de contrôle UPnP sont soient des télécommandes spécialisées pour un type donné ou pour une gamme donnée d'équipements, soient des équipements généralistes disposant d'une IHM (téléphones, PDA ou téléviseur) plus ou moins élaborée. Dans ce deuxième cas, le programme de point de contrôle généraliste doit disposer des composants d'interfaces de contrôle de tous les équipements branchés dans le domicile. Il doit également contenir ceux des équipements à venir. Ces composants peuvent exister par centaines du fait de l'étendue des gammes de

UbiMob'06, September 5-8, 2006, Paris, France.

produits chez les fabricants. De plus, la tâche est compliquée par la co-existence de plusieurs canevas de conception d'IHM. Par exemple, Le développeur Java doit choisir entre plusieurs canevas alternatives pour construire ses IHM: AWT, Swing et SWT pour les stations de travail, eSWT pour les PDA, MIDLet pour les téléphones, HAVi et JavaTV pour les téléviseurs…

De manière similaire, le logiciel des passerelles « micro-mondes » doit lui aussi inclure les composants de conversion permettent de traduire des actions UPnP en commandes réseau vers équipements du « micro-monde ». La passerelle doit donc prendre en compte tout ajout d'un nouvel équipement par la mise à jour de son logiciel d'usine (*firmware*) qui provoque un redémarrage.

Dans ces deux cas, la mise à jour dynamique du programme de points de contrôle et des passerelles micro-mondes n'est pour l'instant pas adressée ni par les produits ni par la littérature.

Dans cet article, nous proposons un canevas de courtage et de déploiement à la demande des composants logiciels destinés aux points de contrôle que nous nommons des *controlet*s et de composants destinés aux passerelles que nous nommons des *bridglet*s. Ce canevas s'appuie sur le paradigme de l'architecture orientée service (SOA) [2]. Le SOA permet de construire des applications dans lesquels toute implémentation d'un service est substituable par une autre pourvu que cette dernière respecte le contrat du service.

La suite de l'article est organisée de la manière suivante : il commence par une présentation des deux types de composants *controlets* et des *bridglets* qui sont associés aux équipements UPnP. La section 3 présente le principe de courtage, de déploiement et de composition de ces composants. La section 4 présente un prototype de ce canevas réalisé au dessus de la plate-forme dynamique de services OSGi [7]. La section 5 positionnera ce canevas par rapport d'autres travaux. Enfin, l'article conclut sur quelques perspectives.

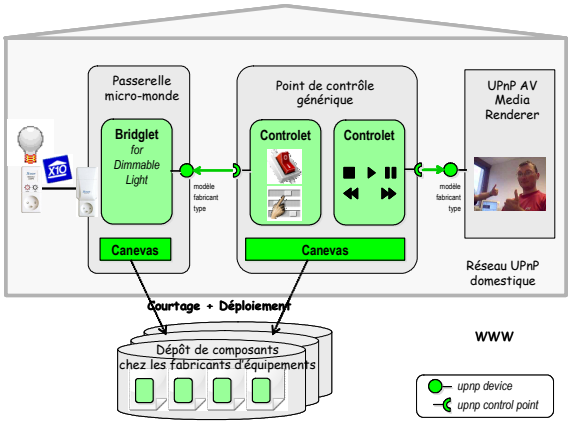

Figure 1 : Bridglets et controlets déployées

2. CONTROLETS ET BRIDGLETS

Dans cet article, nous proposons un canevas de courtage et de déploiement sur demande des composants logiciels destinés aux points de contrôle que nous nommons des *controlet*s et de composants destinés aux passerelles que nous nommons des *bridglet*s (figure 1).

Une *controlet* est un composant de contrôle d'un équipement ou d'un service. Il offre à l'usager une interface graphique pour afficher l'état courant de l'équipement ou du service. Il s'abonne aux variables d'état du service contrôlé afin de rendre son IHM réactive à leurs changements de valeur. Par exemple, le bouton marche/arrêt de la figure 2 bascule seulement quand le changement d'état est notifié. Une *controlet* peut conserver un journal de la liste des actions réalisées par l'usager ainsi que les dernières valeurs connues des variables d'état. La *controlet* peut alors continuer à afficher l'état supposé d'un équipement provisoirement déconnecté du point de contrôle (i.e. perte transitoire du réseau WiFi…).

Une *bridglet* est un composant réalisant des conversions d'opération et de notification entre des équipemens d'un micro-monde et le réseau UPnP. La *brigdlet* présente tout ou partie d'un équipement du micro-monde comme un équipement ou un service UPnP standardisé ou propriétaire. Dans l'exemple de la figure 1, la *bridglet* vers un élément X10 offre un équipement UPnP du type standardisé `DimmableLight`.

Dans ce canevas, les composants *controlets* et *bridglets* ont les propriétés suivantes :

- Modulaire: un composant peut traiter soit un équipement complet, soit un des sous-équipements imbriqués, soit un des services de l'équipement racine ou d'un des sous-équipements. Par exemple, une *controlet* de service (cf bouton *ON/OFF* de la figure 2) peut traiter les opérations marche et arrêt de tout équipement électrique et ainsi être réutilisée par plusieurs *controlets* d'équipements.

- Composable : comme dans de nombreux modèles de composants, un composant peut-être composé lui-même d'autres composants traitant des sous-équipements ou des services. La composition de *controlet* (resp. *bridglet*) suit en général la structure hiérarchique des équipements UPnP. Dans la figure 2, la *controlet* du téléviseur est composée d'une *controlet* marché/arrêt, une *controlet* de sélection des chaînes, et une *controlet* de réglage du volume.

Les sous-composants peuvent être définis statiquement à la conception ou bien être courtés dynamiquement lors de l'utilisation selon l'approche SOA. Dans le cas des *controlets*, un ordre et une priorité d'affichage peuvent être également spécifiés entre les sous-composants. La priorité sert à n'afficher que les *controlets* importantes quand le contexte IHM ne permet que l'affichage d'une partie des *controlets* ou bien quand l'utilisateur souhaite une interface simplifiée. Par exemple, la *controlet* du téléviseur n'affiche pas les *controlets* de réglage du contraste et des couleurs ou bien les extensions *dolby* et 5.1 de la *controlet* relative au volume sonore.

- Polymorphe : le canevas manipule des composants généralistes capables de traiter de manière générique le plus grand nombre d'équipements comme des composants spécialisés traitant uniquement un équipement d'un modèle donné d'un fabricant donné. La spécialisation exploite en général des extensions propriétaires des équipements et de leurs services. Par exemple, une *controlet* spécialisée pour le service de sélection des 99 chaînes d'un téléviseur représentera les 10 touches usuelles d'une télécommande ordinaire tandis qu'une *controlet* générique représentera ce service soit par un champ de texte à remplir soit par une barre de défilement (i.e. *scrollbar*) comportant 99 incréments.

- Substituable et négociable : Conformément aux paradigmes du SOA, plusieurs composants peuvent se substituer les uns aux autres pour offrir un contrôle sur un équipement. Cependant, le canevas recherche toujours en premier celui qui est le plus spécialisé pour l'équipement. Pour cela, chaque composant est attaché à une description listant le modèle et le fabricant de l'équipement traitant, le type de l'équipement et du service traité ainsi que l'environnement d'exécution supporté. La description d'une *controlet* peut-être complétée par la culture (langue de l'usager…). Dans le cas de *controlets*, l'environnement comporte également des indications sur le kit graphique requis.

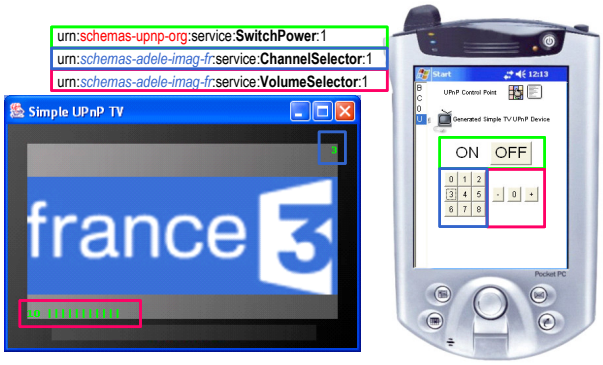

Figure 2 : *Controlet* composite

3. COURTAGE ET DÉPLOIEMENT

Le canevas s'appuie sur les informations attachées aux *controlets* et aux *bridglets* pour réaliser leur courtage, leur composition et leur déploiement.

3.1 Courtage

Pour tout nouvel équipement découvert dans le réseau UPnP (resp. dans le micro-monde), le canevas lance une opération de courtage pour déterminer la *controlet* (resp. la *bridglet*) la plus adéquate pour l'équipement découvert.

Parmi la liste des composants disponibles, l'algorithme de courtage élimine premièrement les composants qui ne satisfont pas les contraintes de l'environnement tels que le toolkit graphique ou bien l'architecture et le système d'exploitation si le composant embarque des bibliothèques natives. L'algorithme recherche ensuite en priorité les composants pour lesquels les attributs modèle et fabricant correspondent pour le type d'équipement ou pour le type de service dans la liste des candidats. Si aucun composant lié au modèle de l'équipement n'est trouvé, l'algorithme recherche alors les composants spécialisés (i.e. sans attributs modèle et fabricant) pour le type d'équipement ou pour le type de service. Enfin, si aucun composant lié au type n'est sélectionné, l'algorithme de courtage retourne par défaut le composant générique. Dans le cas d'un équipement, ce composant agrège les composants pour les sous-équipements et les services de l'équipement découvert. Dans le cas d'un service, ce composant liste les variables d'état et s'abonne à leurs notifications et propose un bouton pour chaque action comme le font les points de contrôle génériques des kits de développement (Intel, Siemens, Domoware).

3.2 Composition

Un composant peut-être lui-même un composite constitué de sous-composants afin d'autoriser des développements modulaires (cf section 2). Ces sous-composants sont liés aux sous-équipements ou aux services d'un équipement. Cependant, il est possible d'envisager d'autres compositions liées, par exemple, aux versions des spécifications de service ou bien par des fonctionnalités étendues par un modèle dans la même gamme d'équipements. Les sous-composants peuvent être définis statiquement à la conception ou bien courtés dynamiquement lors de l'utilisation selon l'approche SOA. Dans ce deuxième cas, le composant utilise à son tour le courtier décrit à la sous-section 4.1 pour rechercher ses sous-composants les plus adéquats. Cette composition par courtage est très similaire à la notion de composant patron (*template*) utilisé dans le modèle Fractal pour définir des assemblages paramétrables de composants.

3.3 Politiques de déploiement

L'opération de déploiement peut-être réalisée immédiatement lorsque l'équipement est découvert ou bien être retardée jusqu'à ce que l'usager commence à réaliser des actions sur l'équipement. La première politique est dite immédiate tandis que la seconde est dite retardée.

La politique immédiate a l'inconvénient d'utiliser les ressources du point de contrôle ou de la passerelle alors que l'usager peut ne pas se servir du tout de l'équipement découvert. Elle peut-être même impraticable lorsque le nombre d'équipements différents s'accroît dans le réseau. Elle a cependant l'avantage sur la politique retardée, de pouvoir notifier l'usager des changements d'état de l'équipement (par exemple, par des messages défilant dans une barre de statut ou de boites d'alerte) dès sa découverte par le point de contrôle.

Une première solution a été d'enrichir statiquement la description de la *controlet* par un attribut définissant la politique à appliquer pour le déploiement du composant. Une seconde solution est de concevoir une *controlet* comme la composition de 2 sous-composants déployés en 2 étapes: l'un chargé des notifications à l'usager et l'autre des actions sur l'équipement. Le premier est déployé selon la politique immédiate tandis que le second selon la politique retardée. Cette dernière solution ne simplifie pas néanmoins le développement et le conditionnement de la *controlet*.

4. PROTOTYPAGE

Ce canevas a été validé par le développement d'un point de contrôle générique pour PDA et d'une passerelle micro-monde pour un bus OneWire™.

Le point de contrôle et la passerelle sont tous deux réalisés en Java au dessus d'une plateforme dynamique de services OSGi [7]. Le choix d'OSGi pour ce prototypage a été motivé à la fois par le fait qu'OSGi permet de concevoir des applications à *plugin* dans lesquels les *plugins* peuvent être déployés et redéployés sans démarrage de l'application principale, mais également par le fait que la spécification OSGi réifie les équipements UPnP sous la forme de services OSGi.

Les informations nécessaires à l'opération de courtage sont regroupées dans un index (grammaire XML) étendant les informations de déploiement utilisées par les services de déploiement d'Oscar et de Felix, 2 implémentations libres d'OSGi. Notre canevas délègue le déploiement au service de déploiement présent sur la plateforme OSGi. Ce dernier déploie la

controlet ou la *bridglet* désirée ainsi que ses dépendances (i.e. d'autres *bundles* fournissant du code ou des services).

Le point de contrôle pour assistant personnel repose un environnement graphique allégé (AWT). Un panneau principal liste les icônes auprès des équipements UPnP découverts. La sélection de l'icône par l'usager provoque le déploiement de *bundle* (i.e. l'unité de déploiement OSGi) choisi par l'opération de courtage. Le *bundle* fournit alors une fabrique d'objets implémentant l'interface *Controlet* pour chaque équipement du même modèle ou d'un même type. La *controlet* pour ce point de contrôle étend la classe *java.awt.Component* ainsi qu'une interface de cycle de vie autorisant l'iconification de la *controlet*. Il a été testé sur HP iPAQ and Dell Axim avec Oscar et Felix sur des JVMs J9 et CReME pour WinCE.

La passerelle micro-monde vers un bus OneWire (http://www.ibutton.com) pilote des capteurs (température, humidité…) qui peuvent être branchés et détectés à chaud sur le bus. Un ensemble de *brigdlets* ont été développé pour une variété de capteurs OneWire. La *bridglet* spécialisée pour les capteurs de température interroge périodiquement ceux-ci selon un mode de *polling* et notifie les changements de valeur des mesures. Dans le cas du OneWire, le courtage de la *bridglet* est réalisé au moyen du modèle ou du type (codé sur 8 bits) du capteur OneWire.

5. - TRAVAUX RELATIFS

Ce travail est à l'intersection des domaines de recherche sur les composants logiciels [8], sur les architectures dynamiques orientées service [2], sur le déploiement des composants logiciels [5] et sur les IHM plastiques [9]. Cette section positionne notre travail par rapport à quelques travaux relativement proches ou apportant des compléments de solutions aux problèmes abordés.

Différentes recherches ont été menées dans le domaine du déploiement de logiciel. Carzaniga et al [5] présente le domaine de déploiement logiciel et liste les principaux critères de comparaison. Dans le monde des applications Java sur terminal, Java Web Start (JNLP) MIDP OTA et OSGi (sur lequel nous appuyons notre prototype) sont parmi les plus avancés. Java Web Start et MIDLet définissent tous deux seulement des dépendances statiques des bibliothèques et d'environnement (i.e. paquetages inclus dans l'environnement) ce qui est inadapté au courtage et au déploiement dynamique des *controlets* et des *bridglets*. Notre algorithme de courtage pourrait être affiné à partir d'informations tirées du contexte ou des contraintes de qualité de services (mémoire utilisée…) comme le proposent les travaux de canevas génériques [1,10].

Les projets Gravity et Beanome [6] explore la création d'IHM dotées de capacités autonomes d'adaptation dynamique en fonction des composants (ressources graphiques) disponible dans le système. Ils s'appuient sur un modèle à composant orienté service simplifiant la tâche du développeur dans la prise en charge du dynamisme des arrivées et départs des composants. Gravity et Beanome n'utilisent cependant que les composants déjà déployés et ils n'adressent pas le courtage dynamique. Les *comets* (COntext of use Mouldable widgETs) [3] sont des composants graphiques adaptant leurs plastiques en fonction du contexte et de l'environnement d'exécution. Elles sont composables comme les *controlets*. Le courtage et le déploiement ne sont pas adressés par les canevas des *Comet*. Les *controlets* pourraient servir de conteneurs de déploiement aux *comets*.

6. CONCLUSION ET PERSPECTIVES.

Cet article a présenté un canevas de courtage et de déploiement sur demande des composants logiciels destinés aux points de contrôle que nous nommons des *controlet*s et de composants destinés aux passerelles que nous nommons des *bridglet*s. Ces composants sont dynamiquement déployés (installés et activés) dès que l'équipement est découvert. Ces composants suivent la structure hiérarchique des équipements UPnP et ils peuvent éventuellement se composer dynamiquement. Un prototypage de ce canevas a été réalisé au dessus de la plateforme dynamique de services OSGi. Des exemples de *controlets* et de *bridglets* illustrant cet article sont accessibles librement depuis http://www-adele.imag.fr/~donsez/dev/osgi.

Une perspective à ce travail est la prise en compte des points de contrôle constitués de composants d'interfaces physiques tels que les *phydgets* pour lesquels notre canevas pourrait servir au déploiement des *phydgetlets* qui seraient des composants logiciels associés aux *phydgets* découvertes dynamiquement.

7. RÉFÉRENCES

[1] Ayed D., Taconet C., Sabri N., Bernard G., Context-Aware Distributed Deployment of Component-Based Applications, *OTM Workshops* (2004)

[2] Bieber G., Carpenter J., Introduction to Service-Oriented Programming, (2001), http://www.openwings.org/

[3] Calvary, G., Coutaz, J., Dâassi, O., Balme, L., Demeure, A. Towards a new generation of widgets for supporting software plasticity: the « comet », *EHCI-DSVIS'2004*, Hamburg, Germany, LNCS 3425 (2004)

[4] Calvary, G., Coutaz, J., Thevenin, D., "A Unifying Reference Framework for the Development of Plastic User Interfaces", *EHCI'2001*, Toronto, LNCS 2254 (2001).

[5] Carzaniga A., Fuggetta A., Hall R.S., Van Der Hoek A., Heimbigner D., Wolf A.L., A Characterization Framework for Software Deployment Technologies. *Technical Report CU-CS-857-98*, University of Colorado, (1998).

[6] Cervantes H, Hall R.S., Automating Service Dependency Management in a Service-Oriented Component Model, *CBSE6 Workshop* Portland, Oregon, USA, (2003)

[7] OSGi Alliance, http://www.osgi.org

[8] Szyperski C., Component Software: Beyond Object-Oriented Programming, *Addison-Welsey* (1997).

[9] Thevenin, D., Coutaz, J., Plasticity of User Interfaces: Framework and Research Agenda. *Interact99*, Edinburgh, A. Sasse & C. Johnson (1999)

[10] Tournier J-C., Olive V., Babau J-P., Qinna, an Component-Based QoS Architecture. *8th SIGSOFT symposium on CBSE*, Saint-Louis, USA, (2005)

[11] UPnP Forum, Understanding UPnP™: A White Paper, Juin 2000, http://www.upnp.org/

[12] Marples D., Moyer S., Home Networking and Appliances, *in Diane Cook, Sajal Das, Smart Environments: Technologies, Protocols and Applications*, Wiley (2004)