

LABORATOIRE



INFORMATIQUE, SIGNAUX ET SYSTÈMES  
DE SOPHIA ANTIPOLIS  
UMR 6070

# COMPARAISON DES SYSTÈMES DE SERVICES POUR DISPOSITIFS

*Vincent Hourdin, Stéphane Lavirotte, Jean-Yves Tigli*

*Projet RAINBOW*

Rapport de recherche  
ISRN I3S/RR-2006-25-FR

Août 2006

# Comparaison des systèmes de services pour dispositifs

Vincent Hourdin, Stéphane Lavirotte, Jean-Yves Tigli

Equipe Rainbow

I3S (Université de Nice – Sophia Antipolis / CNRS) UMR 6070

B.P. 121 – 06903 Sophia Antipolis – France

{Hourdin, Tigli}@polytech.unice.fr

Stephane.Lavirotte@unice.fr

## Résumé

*L'augmentation du nombre d'ordinateurs ainsi que leur miniaturisation et la mobilité des utilisateurs fait naître de nouveaux problèmes liés à leurs interactions. Comment utiliser au mieux les capacités de chaque unité de calcul qui nous entoure ? Comment créer des programmes sensibles au contexte de l'utilisateur ? Comment utiliser les données d'un dispositif sans avoir à écrire de code spécifique pour chaque application ? Les notions de service et de service pour dispositif apportent des réponses à ces questions. Cette étude résume les différents types de services pour dispositifs qui existent, étudie leurs avantages et inconvénients, et cherche à définir un outil mieux adapté à partir de ces enseignements.*

## 1. Architectures Orientées Services (SOA)

### 1.1. Origines et motivations

Les architectures orientées services séparent les données et les traitements, contrairement aux architectures orientées objet. On peut définir un service comme étant un module qui peut être invoqué, qui est assigné à une fonction spécifique, et qui offre une interface bien définie. L'implémentation de la fonction fournie est libre, plusieurs services peuvent la réaliser de différentes manières. Deux types de programmes sont principalement utilisés dans les SOA (*Service Oriented Architecture*): les **fournisseurs** qui mettent à disposition leurs services, et les **consommateurs** qui y font appel.

La vision d'un système destinée à traiter toute application comme un fournisseur de services a de nombreux avantages ; le plus évident est de pouvoir réutiliser un service pour plusieurs applications. Ainsi on peut concevoir les programmes comme des modules. Ils sont indépendants (leur état ne dépend pas des autres) et peuvent être distribués sur différentes machines. Les SOA sont souvent utilisées pour mettre en place des applications

réparties. En utilisant plusieurs fournisseurs en redondance, on accroît la rapidité et la fiabilité d'un système.

Il est possible d'utiliser un logiciel d'orchestration qui crée les liens entre fournisseurs et consommateurs ; les programmes ne sont alors plus statiques. Il est possible de changer de fournisseur pour une fonction précise, éventuellement pendant l'exécution du programme.

Parmi les précurseurs des architectures orientées services, on trouve RPC (Remote Procedure Call), CORBA[1] et DCOM[2], le plus utilisé de nos jours étant probablement RMI[3].

## 1.2. Web Services

Les Web Services sont basés sur les SOA, pour lesquelles ils spécifient un format de représentation de données (XML) et les protocoles de transferts, basés sur les technologies Web recommandées par le W3C. Les principales couches du standard OSI sont ainsi représentées :

- couche réseau : IP
- couche transport : TCP
- couche présentation : XML pour représenter les données. C'est un format de données extensible utilisé assez simplement dans des programmes, des bibliothèques d'analyse étant présentes dans la plupart des langages de programmation. XML est utilisé aussi bien pour la description des services que pour les passages de messages :
  - WSDL : Web Service Description Language, pour décrire les services. WSDL permet de découvrir les services proposés.
  - SOAP : Simple Object Access Protocol, pour invoquer les méthodes sur les Web Services.
- couche application : HTTP. L'utilisation du HTTP sur le port 80 permet de passer facilement les firewalls.

L'utilisation de ces couches standardisées permet de ne pas imposer de langage de programmation lié aux descriptions, contrairement à l'utilisation de souches (*stubs*) ou de code source dans les services classiques. Les couches OSI que nous avons vues sont utilisées par tous les systèmes d'exploitation et architectures. Un Web Service peut donc être écrit pour ou utilisé par n'importe quel type de machine, que ce soit pour les fournisseurs ou les consommateurs. Le travail est tout de même facilité par l'utilisation des SDK de Web Services, par exemple dans les environnements .NET ou Java.

Les Web Services sont référencés par un (ou des) annuaire(s) UDDI, qui connaît la liste de tous les services publics, et leurs fichiers de description WSDL. UDDI fournit trois types de recherches de ces services :

- les pages blanches contiennent des informations sur les services et les entreprises, telles que leur nom, des descriptions textuelles, et les informations de contact de la société,
- les pages jaunes classent les services par catégories. Trois types de taxonomies sont supportés : les codes industriels NAICS, les taxonomies de produits et services UN/SPSC, et les localisations géographiques,
- les pages vertes contiennent des informations techniques sur le moyen d'invoquer les services, et les références vers les spécifications des Web Services.

Les Web Services sont désormais l'exemple le plus courant d'utilisation des architectures orientées services. On les retrouve surtout dans les systèmes de traitement de l'information.

## 2. Services pour dispositifs

Nous avons vu les bénéfices des architectures orientées services pour les programmes classiques. Elles forment une constellation de services, utilisables indépendamment, réalisant chacun un service qui lui est propre. On peut comparer cette vue à celle d'une constellation de dispositifs : chaque dispositif est en général autonome et a des fonctions à offrir. Utiliser des services pour représenter les dispositifs fournit la dynamique nécessaire aux applications mobiles : ils permettent de gérer l'apparition ou la disparition de dispositifs du contexte de l'utilisateur physique comme l'arrivée ou le départ d'un service.

Nous allons étudier en détail les problèmes liés à la communication avec ces services, puis comparer les différents protocoles existants de services pour dispositifs.

### 2.1. Recherche et découverte

La localisation d'un service standard n'a pas d'importance (par exemple un Web Service) ; la seule information nécessaire est la sémantique du service rendu. Pour ce genre de services, on dispose le plus souvent de référentiels d'offres de services, aussi appelés annuaires (ex : UDDI). Ils permettent aux consommateurs d'effectuer une recherche de fournisseurs, en tenant compte de certains paramètres (en général, le type de service, ou son nom). Ils obtiennent alors une référence, qui peut être une adresse Internet dans le cas d'un Web Service ou pour un service classique, du code dit souche (*stub*) qui permet de communiquer directement avec les services, sans avoir à générer de code ou s'occuper de la gestion de la connexion, mais dépendant d'un langage de programmation.

Dans le cas des services pour dispositifs, ceux-ci étant le plus souvent mobiles, leur adresse et leur localisation ne sont pas toujours connues d'un référentiel. De plus, leur localisation peut être un critère de choix pour un type de service, par exemple un capteur de température intérieur ou extérieur.

Pour y accéder, deux moyens sont possibles en général :

- les consommateurs effectuent une méthode de recherche par diffusion (*multicast* ou *broadcast*). Les fournisseurs répondent alors seulement au consommateur (*unicast*) qui a effectué la recherche, en indiquant les informations pour les contacter.
- les fournisseurs annoncent périodiquement leur présence ainsi qu'une date de validité par diffusion. Les consommateurs récupèrent de cette annonce les localisations des fournisseurs.

Les méthodes de diffusion sont valables sur un réseau Ethernet IP grâce à l'UDP, mais la couche de transport ne doit pas être une limite. Avec d'autres protocoles ou médias tels que Bluetooth, WiFi, InfraRouge ou I<sup>2</sup>C, les *multicast* ou *broadcast* sont souvent gourmands en énergie et bande passante. Une solution à ce problème est de séparer la partie transport de la partie applicative, en dédiant une couche de transport optimale à chaque protocole.

Une autre solution, plutôt que de séparer les couches, est de disposer de *proxys* entre les protocoles[6]. Ils permettent d'agrandir le réseau de services, en étendant le réseau classique à celui des protocoles d'ordinaire moins bien supportés. On peut aussi s'en servir comme relais de paquets *multicast* pour les réseaux IP, si on ne se sert pas de répertoire de services sur un réseau de taille moyenne.

On peut donc distinguer deux types de découverte de services, ceux qui sont destinés à être utilisés sur un large réseau ou sur Internet, et ceux destinés à être utilisés sur un réseau local, éventuellement dans une approche contextuelle. Cependant, des dispositifs qui, à première vue, sont utilisés par des utilisateurs locaux, peuvent être aussi utiles à distance. Un capteur de température attaché au réseau d'une maison par exemple, renseignera ses habitants, mais s'ils partent en vacances et veulent connaître la température, il sera nécessaire de disposer d'un répertoire de services local. Les deux approches ne sont donc pas incompatibles, mais complémentaires.

## **2.2. Types de communications**

### **2.2.1. Descriptions**

Parmi les nombreux outils qui existent pour créer des services (pour dispositifs ou non), certains utilisent des protocoles de communication bien connus, tels que WSDL et SOAP, basés sur XML. Récupérer la localisation des services est une chose, la comprendre et communiquer avec en est une autre.

Bien entendu, lorsqu'un client effectue une recherche spécifique par diffusion, seuls les serveurs qui utilisent ce protocole répondront, et logiquement le client sera en mesure de dialoguer avec. Cependant, l'interface<sup>1</sup> du fournisseur n'est pas toujours connue du consommateur, et il doit récupérer une description des services (analogie au WSDL pour les Web services).

Cette description contient une liste de méthodes invocables pour réaliser le service en question, parfois aussi une liste de variables qui renseignent sur l'état du service, et éventuellement une description textuelle des services. Cette interface peut être utilisée, par exemple, pour générer le code qui communiquera avec le fournisseur ou, pour créer un composant réutilisable dans une plate-forme d'assemblage.

### **2.2.2. Messages synchrones : invocations**

Dans l'utilisation d'un service classique, les messages sont synchrones : un consommateur envoie une requête au fournisseur, qui lui retourne un message lorsqu'elle est effectuée. Ce message contient les valeurs de retour c'est à dire en règle générale, les informations demandées par la requête.

Dans les services pour dispositifs, on a souvent besoin d'invoquer des actions pour changer l'état du serveur, et les messages de retour permettent de vérifier que l'action a bien été effectuée. Les dispositifs pouvant être mobiles, il est important de savoir si les fournisseurs visibles sont toujours présents.

---

<sup>1</sup> liste de fonctions des services proposés

### 2.2.3. Messages asynchrones : évènements

Imaginons que nous souhaitions utiliser un interrupteur matériel, et que l'on puisse communiquer avec par l'intermédiaire d'un service. L'utilisation de services pour représenter des dispositifs ne doit pas nous soustraire les fonctionnalités des utilisations plus classiques du matériel. Les interruptions matérielles sont un avantage dont on ne veut pas se passer.

Plutôt que de réaliser une attente active (*polling*) en utilisant les fonctions d'un service, les outils de services pour dispositifs fournissent un système d'abonnement aux évènements. Un client peut à tout moment recevoir —de façon asynchrone— un message du serveur. Ainsi, il recevra un message qui annoncera que l'interrupteur a changé d'état, plutôt que de lui demander périodiquement s'il a changé. Cependant, cette capacité est plus ou moins bien conçue suivant les protocoles. Par exemple avec UPnP, il n'est pas possible de s'abonner à une seule variable, et il faut compter sur le temps d'établissement d'une connexion TCP (tout de même rapide sur réseau local) et le transfert des valeurs de toutes les variables à chaque évènement.

Les abonnements sont en général limités dans le temps. Les réabonnements permettent de vérifier que les consommateurs et fournisseurs sont toujours visibles et actifs. C'est la notion de *leasing*, fortement utilisée dans les services pour dispositifs.

## 2.3. Mise en œuvre : Implémentation

Un problème majeur se pose dans l'implémentation de services pour dispositifs : la plupart des exemples mettent en scène des imprimantes ou des routeurs comme fournisseurs de services, mais rarement des dispositifs mobiles tels que des téléphones portables, des PDA ou des capteurs sans fil. Ces derniers utilisent en général une batterie et ont une capacité de calcul et de mémoire restreintes. Les protocoles de services les plus complexes utilisent le processeur bien plus que si on utilisait les dispositifs par un moyen plus traditionnel, et donc affaiblissent la batterie, et augmentent les temps de latence. Ils nécessitent aussi plus de mémoire, on ne peut pas toujours les embarquer sur le matériel.

Une solution à ce problème est l'utilisation de *bridge* de service. Un bridge est un programme lancé sur un ordinateur qui possède une plus grande puissance de calcul, avec lequel communiquent les dispositifs mobiles *via* leurs protocoles natifs. Il sert d'interface aux services. Ainsi on ne s'adresse plus directement aux dispositifs, mais à un ordinateur, qui permet, de plus, d'en gérer plusieurs et éventuellement modérer les accès.

## 2.4. Sécurité

La sécurité au niveau des Web Services peut se situer à deux niveaux : la sécurité de la découverte des services[7] (accès sécurisé à l'annuaire par exemple), et la sécurité des messages envoyés entre le fournisseurs et le consommateurs pour que ceux-ci ne passent pas en clair sur le réseau.

Toutefois, la plupart des systèmes de services pour dispositifs ne supportent pas l'authentification ou le cryptage. Autrement dit, les invocations effectuées, leurs valeurs de retour, et les évènements passent en clair sur le réseau. On risque donc un espionnage des activités d'un serveur ou d'un client. Si on dispose d'un service qui allume ou éteint la lumière dans une maison, sans authentification, "tout le monde" pourra faire fonctionner le dispositif (y compris vos voisins par exemple)... Cependant, pour certaines applications,

surtout un réseau local ou personnel où la portée des paquets *multicast* reste privée, le cryptage et l'authentification ne sont pas indispensables.

### 3. État de l'art

#### 3.1. Salutation

Salutation a été développé par un consortium industriel ouvert appelé le *Salutation Consortium*. Ce système de découverte de service ne repose pas sur des protocoles, des langages ou des couches physiques définies. On peut l'utiliser sur un réseau TCP/IP aussi bien que sur de l'IrDA ou au-dessus du Bluetooth SDP.

Le cœur de l'architecture de Salutation est le *Salutation Manager* (SLM). Ils permettent aux fournisseurs d'enregistrer leurs services, et de les mettre à disposition des consommateurs. Pour découvrir ou rechercher les services disponibles, les consommateurs effectuent une requête au SLM local, qui, s'il n'est pas capable de répondre, répercutera la requête aux autres SLM du réseau. Idéalement, chaque fournisseur et consommateur dispose d'un SLM local.

L'indépendance de la couche de transport est rendue possible par l'utilisation des *Transport Managers*, qui fournit aux SLM des canaux de communication garantis, quel que soit le protocole de transport de données du réseau.

Les SLM utilisent RPC (*Remote Procedure Call*) pour communiquer entre eux, et fournissent les codes souches ou *stubs* (utilisés pour convertir les types des données locales en types des données d'appels, et transporter les données de manière transparente du entre les deux entités).

Salutation ne possède pas de réel système de notifications d'évènements. En outre, on peut s'abonner au Manager pour connaître périodiquement la disponibilité des services.

Le consortium Salutation n'existe plus. Il est donc probable que Salutation soit de moins en moins utilisé.

**Avantages :** couche de transport séparée, informations sur la charge d'un serveur.

**Inconvénients :** n'est plus développé, plus de site de référence.

**Dates :** juillet 1995 - 30 juin 2005

**Site web du consortium :** -

#### 3.2. SLP : Service Location Protocol

SLP est un protocole de découverte de services développé par le groupe de travail IETF SvrLoc. Son but est de définir un standard indépendant pour les réseaux TCP/IP. Il peut fonctionner aussi bien dans un petit réseau grâce à l'utilisation du *multicast* pour la découverte, que dans un réseau d'entreprise en utilisant les annuaires de services. C'est un des rares protocoles (avec Jini et SCP) de découverte de service qui peut utiliser ces deux modes.

SLP repose sur trois entités : les *User Agents*, qui sont les initiateurs des recherches, les *Service Agents*, qui sont les fournisseurs de services, et les *Directory Agents*, annuaires de services.

Les *User Agents* peuvent effectuer une recherche avec une expression booléenne, par exemple si le schéma de description des imprimantes contient les attributs "q" et "speed",

pour la longueur de la file d'attente et le nombre de page par heure, une *Service Request* peut contenir le prédicat, conforme à la syntaxe LDAP :

( &(q<=3) (speed>=1000) )

Le *User Agent* obtient une URL pointant vers un *Service Agent*, ainsi que quelques informations sur le type de service, le matériel et ses capacités, et la localisation géographique.

Il est possible de renseigner un DHCP avec l'adresse du *Directory Agent*, avec l'option 78. Un client souhaitant obtenir une configuration réseau par DHCP pourrait récupérer en plus l'adresse du DA[9]. De plus, l'annuaire est découpé en *scopes*, qui peuvent avoir chacun une clé secrète pour tous les services qu'ils contiennent, ajoutant ainsi la notion d'authentification et de cryptage de données.

SLP ne définit qu'un protocole de découverte et recherche de services, pas d'accès à ces services, contrairement à Salutation ou Jini par exemple. SLP est implémenté dans de nombreux produits commerciaux, comme ceux de la technologie JetSend de Hewlett Packard. SLP est intégré à Solaris 8. Avant d'utiliser Zeroconf, Mac OS X utilisait SLP pour découvrir les partages de fichiers et autres services. Le serveur d'impression CUPS (Common Unix Printing System) peut utiliser SLP pour s'annoncer sur le réseau.

**Avantages** : indépendant des entreprises, utilisation des deux types de recherche, expressions booléennes pour la recherche, renseignement DHCP.

**Inconvénients** : limité aux réseaux IP

**Dates** : V1 juin 1997 - V2 juin 1999 -

**Site web** : <http://srvloc.sourceforge.net/>

### 3.3. Jini

Jini est une extension de Java développée par Sun Microsystems. Un dispositif qui veut fournir un service Jini doit donc disposer d'une machine virtuelle Java (JVM).

Jini utilise une *Lookup Table* qui est une base de données des services du réseau. En plus de connaître les localisations des services, cette table peut contenir du code Java : cela peut être une interface Java, qui est le format de description de service utilisé ou une classe destinée à être utilisée pour communiquer avec un service, par exemple un pilote ou un programme souche (*stub*) RMI (*Remote Method Invocation*) qui est le protocole d'appel utilisé par Jini. Il est aussi possible, lorsque la localisation de la *Lookup Table* est inconnue, de découvrir les services par diffusion.

De même que les *scopes* de SLP, Jini définit la notion de groupes. Chaque service peut choisir son groupe au moment de l'inscription dans une *Lookup Table*.

Jini fournit une gestion d'évènements, indispensable aux services pour dispositifs, ainsi que la notion de transaction, qui fixe une atomicité aux actions effectuées. Ainsi, lorsque plusieurs clients utilisent un service en même temps, le fournisseur ne peut pas se retrouver dans un état incohérent, qui induirait en erreur les consommateurs.

Bien que la spécification de Jini ne comprenne pas de mécanismes de sécurité, il est possible d'ajouter une couche qui implémente l'authentification ou le cryptage[12].

Il est important de préciser que le J2ME CLDC, machine virtuelle pour les dispositifs de faible puissance de calcul ou de faible capacité, par exemple les téléphones et PDA, ne supporte pas Jini/RMI.

**Avantages :** transactions, indépendant du réseau, groupes, extensible.

**Inconvénients :** dépendant du langage Java et du J2ME, assez lourd à mettre en place.

**Dates :** janvier 1999 -

**Site web du consortium :** <http://www.jini.org/>

### 3.4. Bonjour (Rendezvous)

Bonjour est une implémentation par Apple du standard ZeroConf. Cette architecture permet de découvrir automatiquement les ordinateurs, les équipements et les services dans les réseaux IP de petite taille, sans utiliser de serveurs DHCP ni DNS. Pour cela, il utilise trois autres technologies : *local-link addressing*, MDNS (Multicast DNS) et DNS-SD (*DNS Service Description*). MDNS permet d'assigner un nom d'hôte différent de ceux utilisés par les autres périphériques du réseau, DNS-SD assigne un nom à chaque service et permet aussi de découvrir les services du réseau, en utilisant la diffusion UDP (multicast).

Bonjour est utilisé dans Mac OS X.

Rendezvous a été renommé Bonjour en 2003, suite à une plainte pour violation de marque, Rendezvous étant déjà déposée.

**Avantages :** autonome.

**Inconvénients :** limité aux réseaux IP.

**Dates :** Rendezvous 1999 - Bonjour 2003 -

**Site web :** <http://www.apple.com/macosx/features/bonjour/>

### 3.5. UPnP : Universal Plug and Play

UPnP est développé par l'UPnP Forum fondé par Microsoft, et rejoint par des centaines de sociétés. UPnP est basé sur de nombreuses technologies existantes et fiables. Un nouveau dispositif obtient une IP automatiquement par un DHCP, ou par AutoIP si il n'y en a pas, la découverte est gérée par SSDP (*Simple Service Discovery Protocol*), qui utilise HTTP sur UDP (HTTPU), et sur Multicast UDP (HTTPMU). Les descriptions de dispositifs et services sont en XML, les appels de fonctions se font en SOAP qui utilise XML. UPnP fournit aussi un système d'évènements connu, GENA (*General Event Notification Architecture*), lui aussi basé sur XML et HTTP/TCP.

A l'origine, UPnP disposait d'un système permettant de récupérer les valeurs des variables directement, en SOAP aussi, mais cette fonctionnalité est maintenant déconseillée par le consortium, au profit de l'utilisation de fonctions de get, et peu de serveurs ou clients la supportent.

L'utilisation de toutes ces technologies Web en fait un Web Service pour dispositifs, et c'est le seul connu.

UPnP est aujourd'hui très répandu dans les routeurs, ce qui permet par exemple à Microsoft Windows de configurer le pare-feu du routeur automatiquement pour les transferts de fichiers ou les visioconférences. Les téléviseurs, les lecteurs DVD, et même certains téléphones mobiles de dernière génération sont équipés d'une interface UPnP.

On peut dire qu'UPnP est le système de service pour dispositifs le plus complet, puisqu'il intègre toutes les couches nécessaires, contrairement à SLP ou Bonjour, cependant on peut se demander s'il est adapté aux dispositifs mobiles, qui ont des ressources limitées, vu la complexité de ses protocoles. Ils doivent en effet intégrer, en plus d'une pile TCP/IP,

un analyseur lexical de XML, un serveur HTTP, et un système de construction de paquets de données XML pour répondre aux requêtes et envoyer des évènements.

On préférera donc l'utiliser en tant que passerelle (*bridge*) de services pour dispositifs, rôle dans lequel il convient parfaitement. On notera tout de même l'absence de contrôle d'identité ou de cryptage.

**Avantages :** complet, fiable, page de présentation.

**Inconvénients :** pile assez grosse, limité aux réseaux IP.

**Dates :** 1999 -

**Site web du consortium :** <http://www.upnp.org/>

### 3.6. Bluetooth

La pile Bluetooth dispose du protocole de recherche Bluetooth SDP. Il permet de rechercher une classe de service, des attributs de services, et de naviguer dans les services disponibles. Ce protocole est spécialement conçu pour les dispositifs Bluetooth peu complexes. Il ne s'occupe que du problème primaire de découverte de service et d'établissement de canal de communication. Il ne dispose pas de référentiel d'offre de service, ni de système de notification ou d'abonnement. On n'est pas non plus averti lorsqu'un service devient indisponible. C'est pourquoi on utilise d'autres protocoles de découverte de services au-dessus de Bluetooth SDP, tel que Salutation ou Jini.

Les services découverts sont définis sous forme de *profils*. Ils définissent un type de dispositif, et donc le type de communications qu'il supporte, par exemple le profil série communique avec les commandes ASCII AT.

Le protocole Bluetooth établit un canal de communication RF entre le périphérique maître et le point d'accès, après avoir effectué une authentification par code PIN. Ce dernier peut être crypté lors d'une communication sécurisée.

**Avantages :** Très utilisé, fiable, recherche par classe.

**Inconvénients :** Couche minimale de découverte.

**Dates :** 2001 - V1 novembre 2003 - V2 novembre 2004 -

**Site web du consortium :** <http://www.bluetooth.org/>

### 3.7. Mini

Dans l'approche ubiquitaire, le projet Mini[11] est né de l'idée d'alléger Jini pour pouvoir l'embarquer facilement sur des dispositifs. Mini ne nécessite pas de machine virtuelle Java 2 imposante, une EmbeddedJava ou PersonalJava conforme au JDK1.1.8 convient. RMI n'est pas utilisé, ce qui réduit énormément l'espace mémoire requis pour la pile. Les interfaces java ne sont plus stockées sur une machine tierce, mais sur chacun des serveurs, qui les mettent à disposition par un serveur HTTP. Les clients utilisent la réflexion pour déterminer les méthodes fournies par le service.

Une pile Jini complète avec la JVM nécessite 17.76Mo. Mini n'a besoin que de 1.13Mo, avec Kaffe comme machine virtuelle.

**Avantages :** Légèreté relative, pour les systèmes embarqués.

**Inconvénients :** dépendant du langage Java.

**Dates :** juin 2002 -

**Document relatif :** <http://www.csg.ethz.ch/people/lenders/doa02.pdf>

### 3.8. JESA : Java Enhanced Service Architecture

JESA est une plate-forme de service spécialement conçue pour les dispositifs aux ressources limitées. Le JSDP (*JESA Service Discovery Protocol*) est son système de découverte de services, qui peut utiliser de manière transparente aussi bien le multicast pour ses recherches, qu'un répertoire de services. [5]

Pour les invocations, JESA utilise *NetObjects*. Son utilisation est simple et transparente pour l'utilisateur, les proxies pour accéder aux objets distants sont créés à l'exécution par métamorphose de classe.

Pour être indépendant de la couche transport, une interface de communication JANet (*Java Abstract Network*) a été développée. Cependant, seul IP et le bus CAN sont implémentés, JANet n'est pas encore intégré à JESA, et le projet semble être abandonné.

**Avantages :** Légèreté, utilisation des deux types de recherche.

**Inconvénients :** dépendant de Java

**Dates :** 2002 -

**Site web du fondateur :** <http://wwwiuk.informatik.uni-rostock.de/~spr/jesa/>

### 3.9. SCP : Simple Control Protocol

On ne dispose que de peu d'informations sur SCP, principalement le site Japonais du Fondateur (Microsoft). SCP est un protocole léger de contrôle de dispositifs, indépendant de la couche physique, spécialement conçu pour les dispositifs qui ont peu de ressources en puissance de calcul, mémoire et bande passante. Un bridge SCP-to-UPnP permet de faire participer ces dispositifs à un réseau UPnP. SCP fournit un système de découverte de dispositifs compatibles sur un réseau, des communications *broadcast* et point à point, un système de passage de message géré par une *Property Route*, des formats de descriptions de l'UPnP (XML), un cryptage 128 bits de chaque message, et une gestion de réseaux logiques indépendants sur un même réseau physique.

Les *Property Routes* sont semblables à des liens entre composants. Elles associent deux services entre elles, et lorsque le fournisseur émet un message —de façon asynchrone, il est délivré au consommateur de la *Property Route*.

L'objectif est clair : « *SCP provides a robust, secure, efficient, and royalty-free protocol that has the potential to unify the existing market for home automation* ». Malgré tous ses avantages, SCP ne semble pas obtenir la notoriété escomptée, et est peu utilisé.

**Avantages :** légèreté (sauf XML), protocoles, bridge UPnP, cryptage.

**Inconvénients :** peu d'informations.

**Dates :** 2002 -

**Site web :** <http://www.microsoft.com/japan/windows/scp/>

### 3.10. Résumé

Salutation et SCP sont les deux seuls systèmes de services pour dispositifs indépendants de la couche transport.

Voici un premier tableau récapitulatif des consortiums, activité, et dépendance aux langages et couches de transports des différents systèmes :

	Salutation	SLP	Jini	UPnP	Bonjour	Bluetooth	JESA	SCP
<b>Fondateur</b>			SUN	Microsoft	Apple		Indep.	Microsoft
<b>Consortium</b>	Salutation	SVRLOC	Java	UPnP	Zeroconf	Bluetooth		
<b>Année</b>	1995	1997	1999	1999	1999	2001	2002	2002
<b>Langage</b>	RPC lang	Indep.	Java	Indep.	Indep.	Indep.	Java	Indep.
<b>Transport</b>	Indep.	IP	IP	IP	IP	Bluetooth	IP	Indep.
<b>Activité</b>	Stoppée	Moyenne	Bonne	Bonne	Bonne	Bonne	Faible	Moyenne

Tab. 1: Informations générales

Certains de ces systèmes que nous avons étudié (Bonjour et SLP) ne sont que des protocoles de découverte de services (SDP). Certains utilisent les diffusions de paquets pour atteindre les services qu'ils ne connaissent pas, d'autres demandent à un annuaire, certains supportent les deux. Le tableau suivant indique ces informations, ainsi que le nom des protocoles de découvertes de services :

	Salutation	SLP	Jini	UPnP	Bonjour	Bluetooth	JESA	SCP
<b>Protocol</b>	Manager	SLP	Jini	SSDP	DNS-SD	BT SDP	JSDP	SCP
<b>Diffusion</b>		x	x	x	x	x	x	x
<b>Annuaire</b>	x	x	x				x	x

Tab. 2: Systèmes de découverte

UPnP est le seul Web Service pour dispositif, puisqu'il est le seul à utiliser HTTP pour ses transferts et XML pour la représentation des données. Jini et JESA utilisent des descriptions basées sur le langage Java. Bluetooth utilise des profils matériels fixés par le consortium.

Salutation et Jini permettent de télécharger des *stubs*, codes souche dépendant d'un langage, utilisés pour invoquer les procédures à distance (RPC ou RMI).

	Salutation	SLP	Jini	JESA	UPnP	SCP	Bonjour	Bluetooth
<b>Description</b>	Func. Unit	Types	Java interface		SCPD			Profils
<b>XML</b>						x		
<b>Stubs</b>	RPC		RMI					

Tab. 3: Format des descriptions

Les communications entre fournisseurs et consommateurs sont en général des invocations de méthodes. Certains systèmes fournissent une notification d'évènements (communications asynchrones). La sécurité des données transmises ou visibles peut être un facteur important dans le choix d'un système de services, certains permettent de les crypter ou fournissent une gestion d'authentification :

	Salutation	SLP	Jini	UPnP	JESA	SCP	Bluetooth
Invocation	RPC		RMI	SOAP	NetObjects	Route	Profils
Évènements	i		x	x		x	
Sécurité	Auth	Auth/Crypt	Possible			Crypt	PIN/Crypt

Tab. 4: Communications

## 4. Discussions

### 4.1. Web Service : bonne approche ?

UPnP est le seul véritable Web Service pour dispositifs. Cependant la fiabilité apportée par les protocoles Web n'est pas à l'avantage des piles, à cause de la complexité d'implémentation. De plus, pour un programme embarqué sur un dispositif mobile, un serveur UPnP est parfois trop gourmand en espace mémoire.

Dans la continuité d'UPnP, SCP amène le XML aux dispositifs limités en terme de bande passante et puissance de calcul, à d'autres réseaux qu'IP. Le bridge SCP-UPnP permet d'utiliser ces dispositifs simples sur un réseau IP, de la même manière que les plus complexes en UPnP.

Cependant, on peut se demander si l'utilisation d'un analyseur XML sur des systèmes de faibles capacités est justifiée : elle reste contraignante, tant en taille de code qu'en espace mémoire ou puissance de calcul. Pourquoi alors ne pas utiliser un fichier de description textuel, dans lequel le délimiteur est la fin de ligne, et chaque ligne a une signification bien précise ? Combiné à quelques données de contrôle en binaire (taille des données à transmettre...), on obtiendrait une description simple à exploiter, aussi bien pour un dispositif fournisseur de service que pour un client graphique sur un ordinateur puissant. De même, pour les invocations de méthodes, les envois d'évènements, et éventuellement les récupérations de valeurs de variables, envoyer des données XML pour décrire ce qui est demandé fait perdre beaucoup de bande passante, et augmente la complexité des programmes et le temps de réponse.

Aussi, si on utilise une architecture en bridge, basée sur un PC contrôlant les dispositifs légers ou simplement pour des dispositifs de capacités assez bonnes, le XML ne pose pas de problème, la conception des programmes en est au contraire facilitée. Des analyseurs lexicaux et syntaxiques de XML existent sur la majorité des systèmes d'exploitation, et dans les langages de programmation les plus connus. Le bridge pourrait alors traduire les données XML en ce format simple, et vice et versa pour passer les messages entre les dispositifs légers et un réseau disposant d'une meilleure bande passante.

Pour ce qui est des connexions, sur IP, on dispose de l'UDP pour la diffusion, et du TCP pour le reste. On pourrait ouvrir seulement une connexion persistante pour tout faire, là encore, cela réduirait bien des problèmes (au contraire du HTTP qui crée une connexion TCP à chaque évènement), et diminuerait encore le temps de latence.

### 4.2. Transport

Comme on l'a vu avec SCP et Salutation, ne pas reposer sur une couche transport prédéfinie peut être un grand avantage. Pour des dispositifs mobiles surtout, puisqu'ils ne sont pas destinés à être tout le temps reliés par Ethernet, mais plutôt par WiFi, Bluetooth, infra-rouge, ou toute autre technologie. L'architecture du système de service idéal est donc composée de deux couches : la couche applicative avec l'analyseur lexical simple et la

forge de paquets, et la couche de transport. Il pourrait y avoir plusieurs couches de transport sur le même service, pour être disponible sur plusieurs médias en même temps. Un système de cryptage ou de compression à la volée intégré à la couche transport serait un atout, et étant transparente à l'utilisateur, la conception d'un service en serait inchangée.

### 4.3. Recherche et découverte

Le passage à l'échelle pourrait se faire avec un répertoire de services, comme on l'a vu pour SLP JESA et SCP, qui disposent des deux modes de recherche, réseau local par diffusion et annuaire. L'idéal serait que le répertoire se mette à l'écoute des messages diffusés sur le réseau local, comme un point de contrôle UPnP, pour remplir sa liste de services locaux. Il faudrait qu'il dispose aussi d'un système d'authentification et de cryptage, ce qui permettrait d'utiliser à distance les services d'un réseau local en toute sécurité. Ce répertoire pourrait avoir une interface Web, générée avec la liste de services connus, visible après authentification, pour plus de convivialité, ou pour surveiller l'état du réseau local distant.

## Références

- [1] <http://www.omg.org/corba/>
- [2] <http://www.microsoft.com/com/>
- [3] <http://java.sun.com/products/jdk/rmi/>
- [4] H. Chen, D. Chakraborty, et al., "Service discovery in the future electronic market". In *Proc. Workshop in Knowledge Based Electronic Market*. AAAI, AAAI Press, 2000.
- [5] Stephan Preu. "JESA Service Discovery Protocol". In E. Gregori, M. Conti, A. T. Campbell, G. Omidyar, and M. Zukerman, editors, *Proceedings of Networking 2002*, volume 2345 of LNCS, pages 1196–1201, Pisa, Italy, May 2002. Springer-Verlag. <http://citeseer.ist.psu.edu/preuss02jesa.html>
- [6] Qi He, Dan Muntz. "Multicast Gateway for Service Location in Heterogeneous Ad Hoc Communication", 2002. HP Technical Reports.
- [7] Steven Czerwinski, Ben Y. Zhao, Todd Hodes, Anthony Joseph, and Randy Katz. "An Architecture for a Secure Service Discovery Service". In *Proceedings of MobiCom '99*, Seattle, WA, August 1999. ACM. <http://citeseer.ist.psu.edu/article/czerwinski99architecture.html>
- [8] Marc Haase, Igor Sedov, Stephan Preuss, Clemens Cap, Dirk Timmermann, Time and Energy Efficient Service Discovery in Bluetooth, in *Proceedings of the 57th IEEE Semiannual Vehicular Technology Conference*, Band I, S. 418-422, ISBN: 1090-3038, Jeju, Korea, 2003.
- [9] Charles E. Perkins. "Service Location Protocol for Mobile Users", 1998. Invited Paper, Portable and Indoor Mobile Radio Conference.
- [10] Choonhwa Lee and Sumi Helal. "Protocols for Service Discovery in Dynamic and Mobile Networks". *International Journal of Computer Research* ISSN 1535-6698, Volume 11, Number 1, pp. 1-12. 2002.
- [11] Polly Huang, Vincent Lenders, Philipp Minnig, and Mario Widmer. Mini, "A Minimal Platform Comparable to Jini for Ubiquitous Computing". <http://www.csg.ethz.ch/people/lenders/doa02.pdf>
- [12] Pierre Stadnik. "Étude critique de mécanismes de sécurité pour l'architecture Jini". Mémoire de licence d'informatique, Université Libre de Bruxelles. 2002.