JINI – SOAP :
Présentation

Pierrick Vanneau
Mathieu Samson
Julien Donel

Introduction

La rédaction de ce document a été effectuée à la suite d'un exposé sur les nouvelles technologies réseau SOAP et Jini. L'ensemble des informations présentes dans cet ouvrage a été tiré des normes définies par la W3C en ce qui concerne SOAP et Sun en ce qui concerne Jini. Ce document ne présente que les aspects théoriques du fonctionnement de ces technologies. Il n'est donc pas fait état de l'utilisation au niveau du développement.

Sommaire

I. Service Web

Définition d'un service web, selon la W3C :

Un service Web est un système logiciel identifié par un URI [RFC 2396], dont les interfaces et les attaches publiques sont définies et décrit grâce à une définition XML. Cela peut être découvert par d'autres systèmes logiciels. Ces systèmes peuvent alors agir l'un sur l'autre avec un processus d'enchaînement, en utilisant les messages basés sur XML et véhiculée par des protocoles d'Internet.

II. SOAP

SOAP (Simple Object Access Protocole) est une recommandation du W3C. (World Wide Web Consortium). D'après cette recommandation, SOAP est destiné à être un protocole léger dont le but est d'échanger des informations structurées dans un environnement décentralisé et distribué. SOAP a été implémenté en gardant comme optique deux objectifs principaux : la simplicité et la possibilité d'extension. Pour cela, ce protocole utilise la technologie XML (Extended Markup Language) pour définir un format de message qui puisse être échangé sur différents protocoles de transport.

1. Origines

En 1998 : proposition d'un groupe de sociétés
– Microsoft, DevelopMentor et Userland Software
En 1999 : soumission à l'IETF et émission d'une spécification officielle
– Arrivée d'autres acteurs importants : IBM, Lotus, Compac, IONA, Intel…
En 2001 : Proposition de SOAP 1.1
– 8 mai 2001 : Diffusion des spécifications de SOAP 1.1 par le W3C
– 17 décembre 2001 : SOAP 1.2 Working draft du W3C

2. Principes de SOAP

SOAP codifie simplement une pratique existante, à savoir l'utilisation conjointe de XML et http. SOAP est un protocole minimal pour appeler des méthodes sur des serveurs, services, composants, objets.

Il a pour avantages de ne pas imposer une API ou un runtime, ni l'utilisation d'un ORB (CORBA, DCOM, …) ou d'un serveur web particulier (Apache, IIS, …) et non plus de ne pas imposer un modèle de programmation.

Une des volontés du W3C vis à vis de SOAP est de "ne pas réinventer une nouvelle technologie". SOAP a été construit pour pouvoir être aisément porté sur toutes les plates-formes et les technologies

SOAP peut être vu selon deux aspects :

- SOAP peut être vu comme un protocole d'échange de "message" :

La requête contient un seul message (appel sérialisé d'une méthode sur un objet) ;
La réponse contient un seul message (retour sérialisé d'un appel de méthode sur un objet) ;

- SOAP peut être vu comme un format d'échange de documents :

La requête contient un document XML.
Le serveur retourne une version transformée

L'architecture client serveur peut être représentée de la manière suivante :

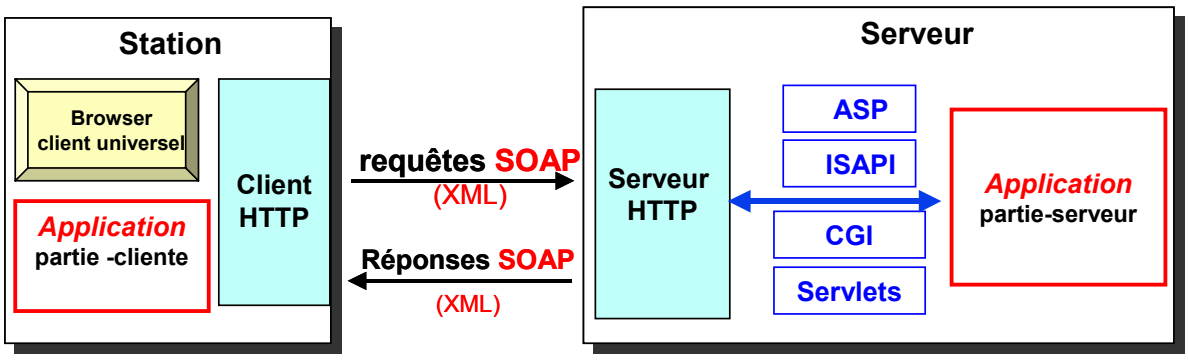


Figure 1 : Utilisation de SOAP dans une architecture client serveur

3. Modèle d'échange de message SOAP

En dehors du protocole auquel SOAP est attaché, les messages sont routés sur un chemin appelé un 'message path', qui permet de transmettre un message à un destinataire en passant par un ou plusieurs noeuds intermédiaires.

Une application SOAP recevant un message SOAP doit traiter ce message en faisant les actions suivantes :

- Identifier toutes les parties du message SOAP ;
- Vérifier que toutes les parties mandataires sont supportées par l'application pour ce message et les traiter en conséquence. Si ce n'est pas le cas le message est supprimé ;
- Si l'application SOAP n'est pas la destination finale du message alors le message est forwardé.

4. Relation avec le XML

Tous les messages SOAP sont encodés en XML. Une application SOAP doit inclure le namespace SOAP adéquat pour tous les éléments et les attributs définis par SOAP. Elle doit pouvoir supprimer les messages qui ont un namespace incorrect.

SOAP définit deux namespaces :

- L'enveloppe SOAP qui a l'identifiant suivant : http://schemas.xmlsoap.org/soap/envelope/
- Les règles d'encodage qui a l'identifiant suivant : "http://schemas.xmlsoap.org/soap/encoding/"

Une message SOAP ne doit pas contenir de DTD.

5. Format d'un message SOAP

Un message SOAP est formé de trois éléments :

- L'enveloppe SOAP (SOAP envelope) définit une cadre d'ensemble pour décrire ce qui est dans le message, qui gère ce message et si ce message est optionnel ou mandataire ;
- Les règles d'encodage définissent un mécanisme de sérialisation qui peut être utilisé pour échanger des types de données d'une application précise ;
- Le SOAP RPC qui définit une convention qui peut être utilisé pour représenter des appels et réponses de procédures distantes.

Bien que ces trois parties soient décrites ensemble en tant que des parties de SOAP, elles sont fonctionnellement différentes. En particulier, l'enveloppe et les règles d'encodage sont définies dans des espaces de nommage (namespaces) différents pour faciliter la modularité.

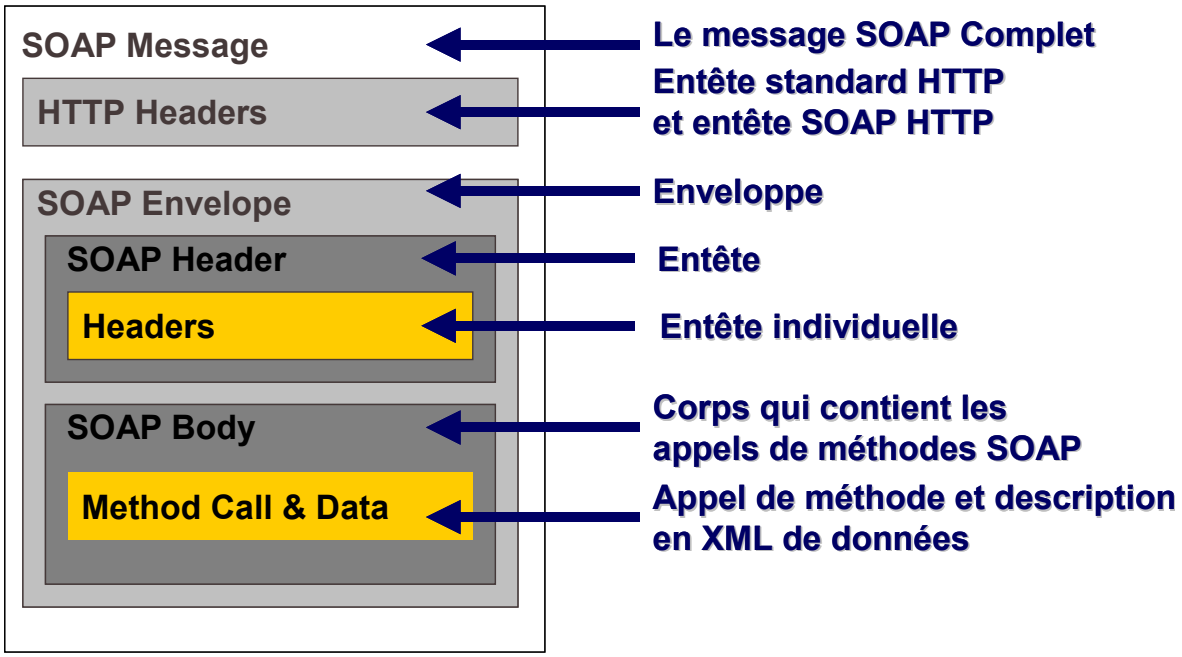


Figure 2 : Format d'un message SOAP

6. L'enveloppe SOAP

Un message SOAP est un document XML qui consiste en une enveloppe mandataire SOAP, un header optionnel et un body. Un message SOAP contient :

- L'enveloppe est l'élément de base du document XML, qui représente le message ;
- Le header est un mécanisme générique qui permet d'ajouter des fonctionnalités à un message SOAP de manière décentralisé sans agrément entre les parties qui communiquent. C'est ici qu'il est indiqué si le message est mandataire ou optionnel ;
- Le body est un container pour les informations mandataires à l'intention du récepteur du message. SOAP définit un élément pour le body, qui est l'élément 'Fault' utilisé pour reporter les erreurs.

Les règles de grammaires sont les suivantes :

- Enveloppe :
  i. Le nom de l'élément est 'envelope' ;
  ii. L'élément doit être présent dans un message SOAP ;
  iii. L'élément peut contenir une déclaration de namespace et des attributs additionnels.
- En-tête (header) :
  i. Le nom de l'élément est 'Header' ;
  ii. L'élément peut être présent dans un message SOAP. S'il est présent, il doit être le premier enfant d'un élément SOAP envelope ;
  iii. L'élément peut contenir une collection d'en-têtes qui doivent être ses enfants immédiats.
- Corps (body) :
  i. Le nom de l'élément est 'Body' ;
  ii. L'élément doit être présent dans un message SOAP et doit être un enfant immédiat d'une envelope SOAP. Il doit suivre le header s'il est présent. Sinon il doit être le fils immédiat de l'enveloppe ;
  iii. L'élément peut contenir une collection de body qui sont tous les fils immédiats d'un body.

7. L'en-tête SOAP

SOAP fournit un mécanisme permettant d 'étendre un message d'une manière modulaire et décentralisée sans mécanisme d'apprentissage entre les deux parties communicantes. Des exemples d'extension qui peuvent être implémentées comme des en-têtes sont des authentifications, des transactions, des paiements…

L'élément Header est encodé en tant que premier enfant de l'enveloppe SOAP. Tous les enfants immédiats du header sont appelés des entrés du header.

Les règles d'encodage pour les entrées du header sont les suivantes :

- Une entrée d'header est identifiée par sont nom d'élément, qui consiste en le namespace URI et le nom local ;
- L'attribut SOAP : 'encodingStyle' peut être utilisé pour indiquer le style de codage de l'entrée ;
- L'attribut SOAP : 'mustUnderstand' peut être utilize pour indiquer comment traiter l'entrée.

8. Le corps du message SOAP (Body)

Le corps d'un message SOAP fournit un mécanisme pour échanger les informations mandataires à l'attention du récepteur final du message. L'utilisation typique de l'élément body inclut les appels RPC et les rapports d'erreurs.

L'élément body est encodé en tant que fils direct de l'élément SOAP envelope. Si un header est présent alors le body doit être le fils immédiat de celui-ci.

Tous les enfants immédiats de l'élément body sont appelés des entrées du corps et chaque entrée est encodée comme un élément indépendant dans l'élément body.

Les règles d'encodage des entrées du body sont les suivantes :

- Une entrée body est identifiée par son nom d'élément, qui correspond au namespace URI et au nom local ;
- L'attribut SOAP 'encodingStyle' peut être utilisé pour indiquer le style d'encodage pour l'entrée.

SOAP définit une entrée body, qui est l'entrée 'Fault' utilisée pour le rapport d'erreurs.

Les éléments ainsi véhiculés peuvent contenir deux types de donnée :

- les types simples

Type	Example
int	58502
float	314159265358979E+1
negativeInteger	-32768
string	Louis "Satchmo" Armstrong

La déclaration d'un type simple s'effectue en deux temps :

1. Il est nécessaire de fixer le type ainsi que le nom de la variable que l'on souhaite manipulée.
2. Comme pour tout langage, nous pouvons ensuite affecté la variable à une valeur.

Exemple :

```
<element name="age" type="int"/>
<element name="height" type="float"/>
<element name="displacement" type="negativeInteger"/>

<age>45</age>
<height>5.9</height>
<displacement>-450</displacement>
```

- Les énumérations

Contrairement aux éléments simples, il faut trois étapes pour utiliser une énumération :

1. Il faut déclarer un nouvel élément qui possède le type que l'on s'apprête à créer
2. Ensuite, on définit les valeurs de l'énumération
3. On utilise l'énumération en affectant la valeur souhaitée.

Exemple :

```
<element name="EyeColor" type="tns:EyeColor"/>
<simpleType name="EyeColor" base="xsd:string">
  <enumeration value="Green"/>
  <enumeration value="Blue"/>
  <enumeration value="Brown"/>
</simpleType>

<Person>
  <Name>Henry Ford</Name>
  <Age>32</Age>
  <EyeColor>Brown</EyeColor>
</Person>
```

Alors que le header et le body sont définis comme des éléments indépendants, ils sont en fait liés. La relation entre une en-tête et un corps est la suivante : un body est sémantiquement équivalent à un header sans destinataire et avec un message SOAP 'mustUnderstand' avec une valeur de '1'.

9. Message 'Fault'

L'élément Fault de SOAP est utilisé pour acheminer des erreurs ou des informations sur le statut à travers un message SOAP. S'il est présent, l'élément Fault de SOAP doit impérativement apparaître comme une entrée du 'body' et ne doit en aucun cas apparaître plus d'une fois à l'intérieur de l'élément 'body'.

L'élément Fault de SOAP est défini par les quatre sous éléments suivants :

faultcode

L'élément faultcode est prévu pour être interprété par des logiciels possédant un algorithme d'identification des erreurs SOAP. Le faultcode doit absolument être présent dans l'élément fault de SOAP. La valeur du faultcode est définie par la norme W3C.

faultstring

L'élément faultstring offre une explication lisible pour l'homme de erreur. Cet élément n'est donc pas interprété par un processus de gestion d'erreur SOAP. Cet élément peut être assimiler au "Reason Phrase" définie par HTTP. Il doit absolument être présent dans l'élément fault de SOAP.

faultactor

L'élément faultactor est prévu pour informer le processus responsable de l'erreur. La valeur de cet élément SOAP est une URI afin d'identifier la source. Toutes les applications qui ne connaissent pas la destination finale d'un message SOAP doivent inclure cet élément.

detail

L'élément détail est prévu pour indiquer le champ d'action d'une erreur qui porte sur les données présentes dans l'élément body.

L'absence d'élément detail dans un message fault signifie que l'erreur n'influe pas sur le traitement des informations contenues dans l'élément Body.

Tous les fils de l'élément detail (au sens xml du terme) sont appelés "detail entries". Chaque "detail entries" apporte une information sur l'élément detail.

Les règles de codage des “detail entries” s’effectue comme suit:

1. Un “detail entries” est défini par son nom.
2. Un attribut “encodingStyle” qui doit être utilisé pour indiquer la manière dont il est codé.

Chaque « detail entries » contient des sous éléments. Ces sous éléments sont différents suivant le type de « detail entry ».

III. JINI

a .Introduction

Jini est un type d'environnement qui offre une implémentation « plug and play » réseau pour des services. Cela signifie qu'un dispositif ou un service de logiciel peut être relié à un réseau et annoncer sa présence. Les clients qui souhaitent employer un tel service peuvent alors le localiser et l'appeler pour résoudre des tâches.

Jini peut être employé pour des tâches de calcul à distance (comme SOAP). De plus, un service peut être relié à un réseau pendant une période courte seulement. Il peut, plus généralement, être employé dans n'importe quel réseau où il y a un certain degré de changement.

Il y a un grand nombre de scénarios qui mette en application ceci :

1. Une nouvelle imprimante peut être reliée au réseau et annoncée sa présence et ses fonctions. Un client peut alors utiliser cette imprimante sans avoir à être particulièrement configuré pour l'utiliser.

2. Un appareil-photo numérique peut être relié au réseau et présenté une interface utilisateur qui permettra non seulement la prise de photo, mais aussi la possibilité rendre compte de sa présence à toutes les imprimantes de sorte que les impressions des photos s'effectuent en même temps.

3. Un dossier de données partagé qui est copié et modifié sur différentes machines peut être transformé en un service de réseau à partir d'une simple machine, ce qui réduit des coûts d'entretien.

4. De nouvelles fonctionnalités peuvent être ajoutées à un système existant sans perturbation, sans besoin de modifier les clients.

5. Les services peuvent annoncer des changements d'état, comme par exemple lorsqu'une imprimante manque de papier. De ce fait, les personnes en charge des consommables pourront agir.

Jini n'est pas un acronyme, et n'a pas une signification particulière. Un système ou une fédération de Jini est une collection de clients qui profite de services via les protocoles de Jini.

La plupart du temps, ceci se compose d'applications écrites en Java, communiquant en utilisant un mécanisme d'invocation de méthode Java à distance. Bien que Jini ne soit écrit qu'en Java pur, ni des clients ni les services sont contraints d'être en Java.

Quand vous téléchargez une version de Jini, vous obtenez plusieurs choses :

Premièrement, Jini contient des spécifications sur un ensemble de composants logiciel. Ceci inclut une api de sorte que vous puissiez écrire vos propres services en tant que programmeur.

Deuxièmement, il inclut une implémentation (en Java pur) du middleware, présenté comme ensemble de paquets de Java.
En incluant ces derniers dans le classpath de votre client ou de votre service vous pouvez profiter des protocoles logiciels de Jini, comme n'importe quelle utilisation d'un jar java.

Enfin, vous obtenez le code source des librairies jini.

En conclusion, Jini exige un certain nombre de services standard. Sun donne d'ailleurs des réalisations de base pour chacun de ces derniers. Ces réalisations ne sont pas des parties officielles de Jini, mais sont incluses pour vous puissiez commencer vos développement.

Dans la pratique, la plupart des utilisateurs trouvent ces derniers suffisamment pour effectuer un développement avec Jini.

b. Les composants

Jini est assimilable à un grand nombre d'architectures de systèmes distribuées, y compris les systèmes industriels dominants tel que CORBA. Sa particularité provient de son implémentation purement en java. Il existe d'ailleurs une possible communication entre les deux mondes CORBA et Jini.

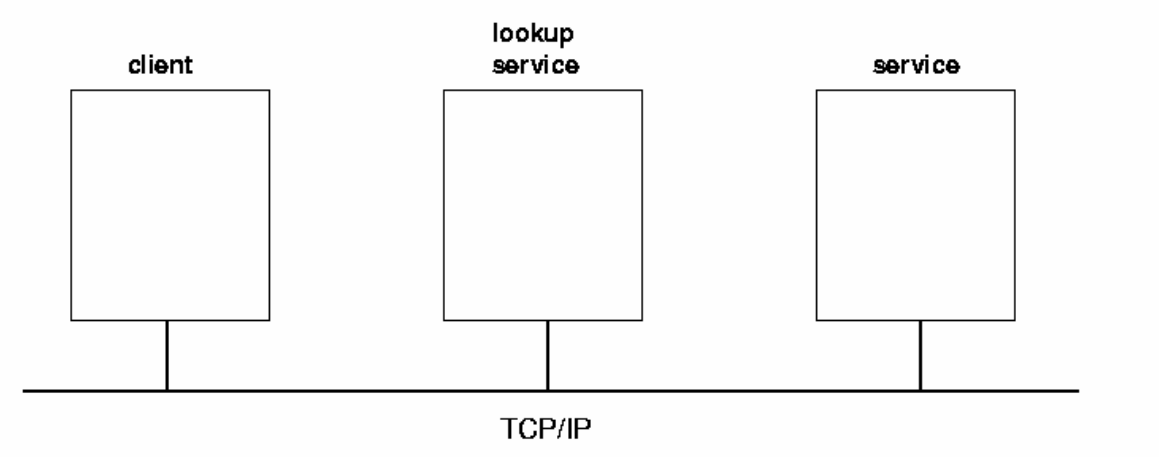
De plus, il existe d'autres frameWorks Java, développé par Sun qui semble être complémentaire à Jini, tel que des EJBs. L'EJB facilite la construction des serveurs logique pour le buisness, et Jini peut être employé pour distribuer ces services de façon de « network plug and play ».

Un utilisateur peut se rendre compte que Jini n'est pas la seule architecture de fournisseur de service sur le marché. Ce qui conditionnera le succès ou l'échec de Jini sera en partie la politique de Sun, mais également (si tout va bien!) les possibilités d'évolution techniques de Jini.

Dans un système courant de Jini, il y a en effet que trois acteurs principaux.

- Un service, tel qu'un imprimeur, un grille-pain, une agence de mariage, etc.
- Un client qui voudrait se servir de ce service.
- Un service de consultation (lookup) qui agit en tant qu'annuaire de service. Cette consultation fait le lien entre les services et les clients.

Il existe cependant un composant additionnel : un réseau reliant chacun ce ces trois acteurs. Ce réseau est généralement TCP/IP (les spécifications de Jini sont assez indépendantes du protocole de réseau, mais l'exécution courante est sur TCP/IP.)



C .Le service d'enregistrement

Un service est un concept logique tel qu'un service de chat, un disque. Il s'avérera être habituellement défini par une interface de Java.

Chaque service peut être développé de beaucoup de manières possibles et par beaucoup fournisseurs différents.

Un service est créé par un fournisseur de service. Un fournisseur de service joue un certain nombre de rôles:

1. Il crée les objets qui mettent en application le service

2. Il enregistre un de ces derniers dans le lookup. L'objet de service est la partie publique et sera téléchargé par les clients.

Pour que le fournisseur de service enregistre l'objet de service dans le lookup, le serveur doit d'abord trouver le serveur de lookup. Ceci peut s'effectuer de deux manières:

1. si l'hébergement du service de consultation est connu, le fournisseur de service peut employer l'unicast TCP pour se relier directement à lui.

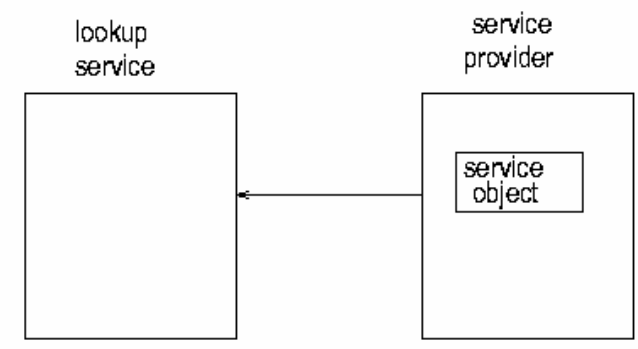
2. Si l'hébergement n'est pas connu, le fournisseur de service fera des demandes de multicast de UDP pour que les services de consultation puissent répondre à ces demandes.

Les services de consultation (lookup) écouteront sur le port 4160 l'unicast TCP et les demandes de multicast.

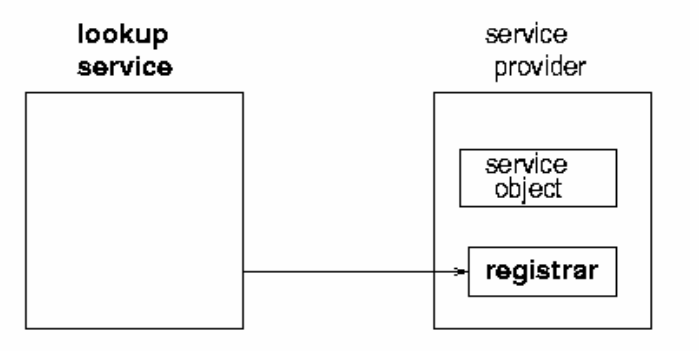
Quand le service de consultation obtient une demande sur ce port, il envoie l'objet au serveur. Cet objet, connu sous le nom de conservateur (*registrar)*, agit en tant que proxy au service de consultation, et fonctionne grâce à la JVM du service. Toutes les demandes que le fournisseur de service doit faire du service de consultation sont faites par ce proxy.

N'importe quel protocole approprié peut être employé pour faire ceci, mais dans la pratique les réalisations qui vous obtenez du service de consultation (par exemple Sun) emploient le RMI. Ceci implique de prendre une copie de l'objet de service, et de la stocker sur le lookup.

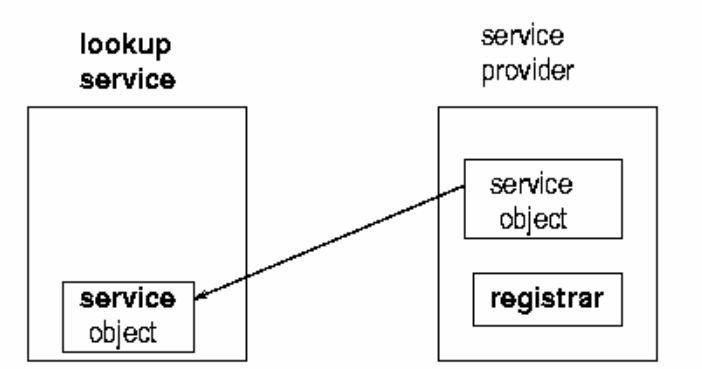
Requête pour l'annuaire de service :



Enregistrement est retourné :



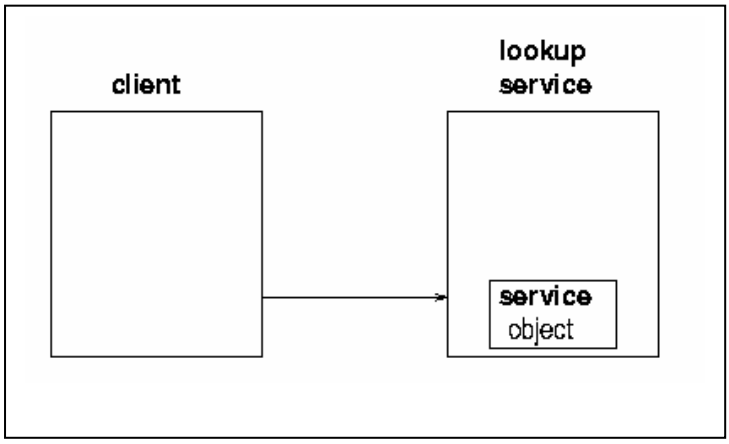
Le téléchargement du service:



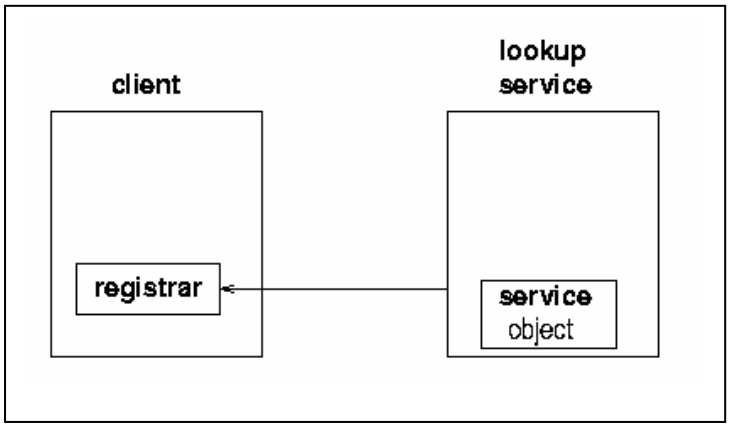
d. le client lookup

Le client du lookup essaye d'entrer une copie du service dans sa propre JVM. Il passe par le même mécanisme pour obtenir un conservateur du lookup. Mais cette fois il fait quelque chose de différent : il doit télécharger l'objet de service.

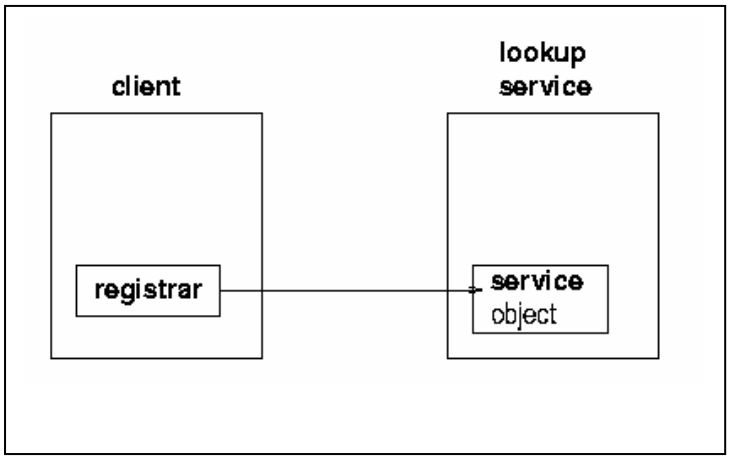
Requête pour l'annuaire de service :



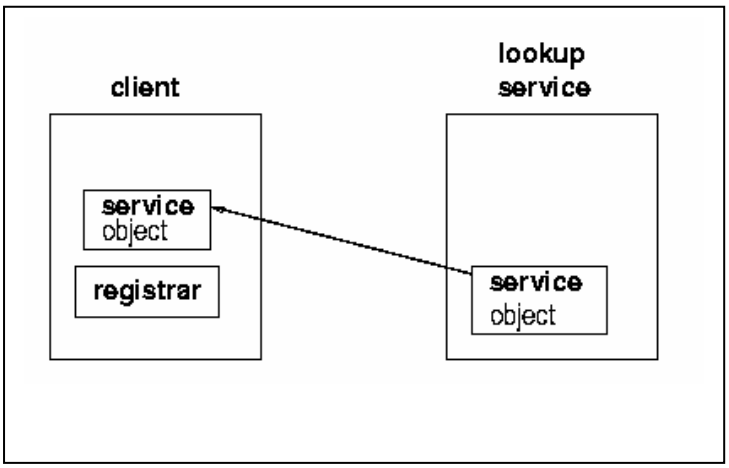
Enregistrement est retourné :



Demande du service :



Retour du service :



e. le proxy

Quelques services peuvent être mis en application par un simple objet, l'objet de service.

Comment va-t-il effectuer un travail adapté si le service est un grille-pain, un imprimeur ?

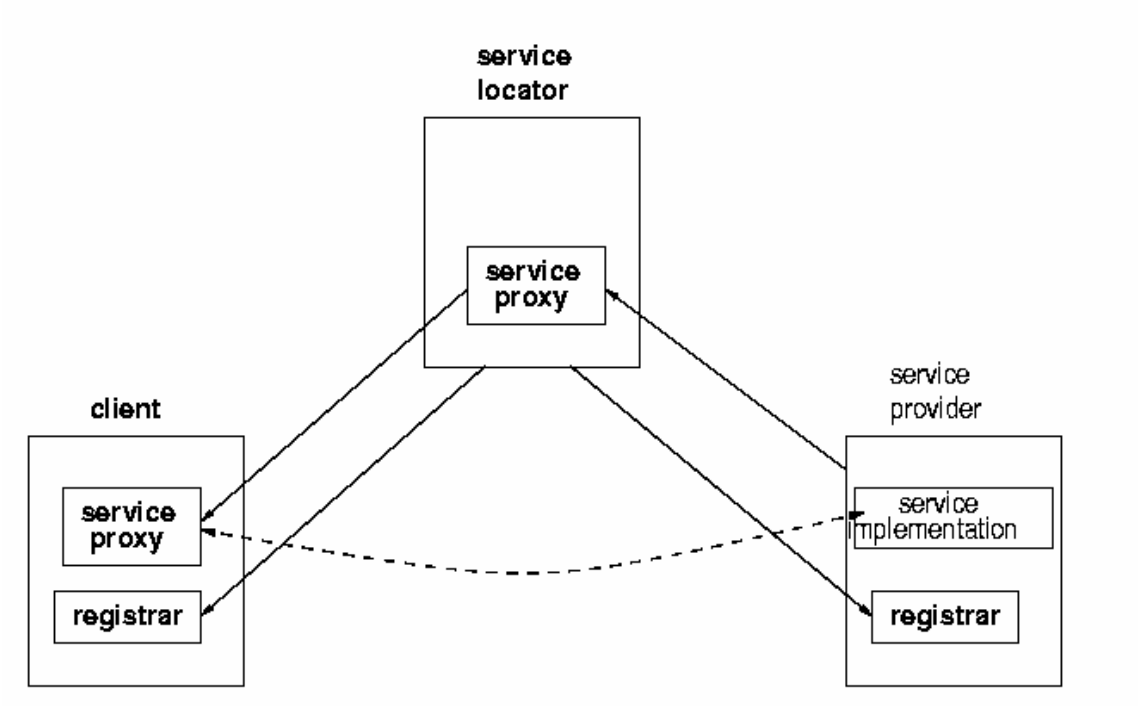
Il ne peut pas commander ce matériel seul. Dans ce cas-ci, l'exécution du service doit se composer au moins de deux objets, un pour le fonctionnement sur client et un pour le fonctionnement sur le fournisseur de service. L'objet de service est vraiment un proxy, qui communiquera avec d'autres objets sur le fournisseur de service, employant probablement le RMI.

Le proxy est la partie du service qui est évidente pour les clients. Sa fonction est de passer des appels de méthode au reste des objets qui forment toute l'exécution du service.
Il n'y a pas une nomenclature standard pour ces objets.

L'intérêt d'utiliser un proxy intervient lorsque un objet de service doit commander une partie matérielle qui n'est pas accessible directement par l'objet de service.
Cependant, le problème peut ne pas être matériel : il pourrait y avoir des dossiers accessibles par fournisseur de service qui ne sont pas disponibles aux objets fonctionnant sur les clients.

Il pourrait y avoir des applications locales au fournisseur de service qui ne sont utiles que lors de la mise en fonctionnement du service.

Le proxy :



Comment le proxy s'amorce-t-il? Ceci n'est pas indiqué par Jini, et peut être fait d'un grand nombre de manières. Par exemple, le RMI qui appelle le service peut être employé comme rmiregistry, où le proxy donne le nom du service.

Ce n'est pas très commun, car des proxys de RMI peuvent être passés directement comme valeur retournée des appels de méthode.

Le proxy peut être aussi mis en application sans l'utilisation directe de RMI, et peut alors employer un autre protocole comme le ftp, le HTTP.

f. Architecture des clients JINI

Machine virtuelle Java individuelle

La méthode la plus directe pour intégrer simplement un microprocesseur, de mémoire vive, de capacité de stockage, bref d'une machine virtuelle Java. Il n'est bien sûr pas nécessaire que cette JVM soit complète : l'équipement client se passera très bien des classes graphiques... Les fonctionnalités requises d'une telle JVM " allégée " sont la capacité de télécharger du code (en fait tout ce qui a trait aux fonctions réseau et sécurité) et de traiter les classes des *packages* spécifiques à Jini. Le contenu exact d'une JVM *light* n'a pas été spécifié par les concepteurs de Jini.

Cette approche présente, malgré ou en raison de sa simplicité, un défaut majeur : le coût de l'équipement client est largement accru par l'addition de la machine virtuelle Java, même dans sa version allégée. En revanche, la flexibilité offerte par la machine virtuelle permet une évolution très simple des fonctionnalités offertes par le service : il suffit au constructeur de fournir une nouvelle version du code Java représentant le service, sans avoir besoin de recourir à une modification matérielle plus coûteuse.

Machine virtuelle spécialisée

L'équipement client peut voir son coût de production baisser si son constructeur renonce à la doter d'une véritable JVM ou d'une JVM allégée, et en remplaçant celle-ci par une machine virtuelle spécialisée qui implémente de façon propriétaire les interfaces avec les services constitutifs Jini (*lookup*, *discovery/join*, etc.). En échange de cette réduction des coûts, on perd un peu de la flexibilité offerte par la solution " 100% pur Java ".

Machine virtuelle Java partagée

Toujours dans une optique de réduction des coûts, on peut mettre en place une machine virtuelle Java complète, partagée entre un ensemble de dispositifs. La communication entre les dispositifs et la JVM est laissée à la discrétion du constructeur (protocoles propriétaires). L'équipement client pourrait partager avec le four à micro-ondes, le lave-vaisselle et le mixer une JVM " de cuisine ", faisant office de proxy au sein de la fédération Jini.

La baisse de coût induite est notable, mais implique une plus grande complexité de la gestion des communications entre le proxy et les

appareils. Le protocole doit être défini à l'avance et ne peut guère évoluer avec autant de souplesse que dans les solutions précédentes.

Jini et Internet Inter-Operability Protocol

Jini peut intégrer des services fondés sur une architecture CORBA à l'aide du protocole IIOP, implémenté dans le package Java RMI (*Remote Method Invocation*).

Si l'équipement client contient un ORB (*Object Request Broker*) CORBA, il peut directement interagir avec le service de recherche Jini, via le protocole IIOP. Le fait que le service de recherche génère ses flux IIOP via RMI est transparent pour le service, et réciproquement le fait que le service n'implémente pas RMI (par exemple pour les questions de *leasing*) l'est pour le service de recherche.

L'équipement client peut également ne pas intégrer d'ORB, mais interpréter directement le flux IIOP et tenter d'interagir avec le service de recherche. Dans ce cas, l'implémentation réside de manière très étroite sur la spécification, et sa compatibilité avec les versions futures de Jini n'est pas garantie, notamment en ce qui concerne le service d'allocation de ressources.

g. Avantages et Inconvénients

Jini est une initiative de Sun Microsystems, qui tente de généraliser l'application de Java. Cette technologie possède donc l'avantage de posséder un soutien industriel fort puisque Sun est un acteur important du marché informatique. Cependant, même si Jini s'attirait le soutien de l'ensemble des acteurs du monde de l'informatique et des réseaux, l'ambition de Sun de généraliser Jini à la domotique, comme dans l'exemple de la cafetière, nécessiterait de convaincre les constructeurs électroménagers, ce qui est loin d'être acquis.

Cependant, si ses ambitions semblent un petit peu irréalistes, Jini n'en demeure pas moins un concept très intéressant dans le cadre de l'informatique des systèmes distribués ou de l'architecture des systèmes. La banalisation de la notion de service et l'unification de l'interface sont deux points forts importants de Jini, qui seront sans doute repris. De plus, le modèle Jini constitue un cadre de pensée structuré et propre, et le système peut être caractérisé par sa grande fiabilité.

Toutefois, Jini a les faiblesses de Java, sur lequel il s'appuie : il requiert de gros moyens en mémoire ou en CPU par exemple, et peut donc se révéler assez lent. En fait Jini, en se concentrant sur des concepts d'assez haut niveau, se détache des problèmes d'optimisation. De plus, l'insertion et l'utilisation de JVM dans les appareils se révélera certainement coûteuse et contraignante. Enfin, les concepteurs de Jini ont tablé sur un environnement où les réseaux à hauts débits sont généralisés, ce qui constitue une anticipation assez marquée.

On peut donc dire que Jini est une très bonne vitrine des capacités de Sun à tirer le meilleur profit de Java. Et même si l'application de Jini à la domotique semble un peu futuriste, on peut penser que certains des concepts de Jini seront reconnus et réutilisés.

IV. Bibliographie

Voici la liste des sites qui nous ont permis d'effectuer ce document ainsi que l'exposé :

Pour SOAP :

- http://www.w3.org/TR/soap/
- http://www.w3schools.com/soap/soap_fault.asp

Pour Jini :

- http://www.sun.com/software/jini/
- http://www.jini.org/