Approche pour un chargement contextuel de services sur des dispositifs contraints

Amira Ben Hamida [*] [**] –Frédéric Le Mouël [*] –Stéphane Frénot [*] – Mohamed Ben Ahmed [**]

[*] *INRIA ARES / CITI, INSA-Lyon, F-69621, France*
[**] *RIADI, ENSI-Tunis, T-2010, Tunisie*
{amira.ben-hamida, frederic.le-mouel, stephane.frenot}@insa-lyon.fr,
{mohamed.benahmed}@riadi.rnu.tn

RÉSUMÉ. *L'informatique ubiquitaire décrit un scénario où les applications sont à la disposition des utilisateurs, n'importe quand et à n'importe quel moment à travers les périphériques mobiles. Un tel scénario reste pour le moment difficile à réaliser. Des efforts devrant être déployés afin d'adapter les applications aux dispositifs mobiles contraints. Les travaux existants abordent les problèmes des ressources contraints et ommettent de considérer la conscience du contexte, ou vice versa. Dans cet article, nous nous basons sur le paradigme de service et proposons une architecture orientée-service qui fournit un chargement optimisé de services dans des environnements pervasifs. Des descriptions contextuelles des services, périphériques et profils utilisateurs sont pris en compte pour atteindre une meilleure adaptation aux contraintes.*

ABSTRACT. *In the vision of ubiquitous computing, applications may be accessed everywhere and at any time through devices. However, such scenario is difficult to realize, yet. Efforts should be made to adapt applications to mobile and constraint devices. Existant work address resources constraint devices but do not consider context-awarness and vice versa. In this paper, we consider the service paradigm and propose a service-oriented architecture providing an optimized services loading in pervasive environments. Contextual descriptions of services, devices and users profiles are taken into account to achieve a better adaptation to constraints.*

MOTS-CLÉS : *architecture orientée services, chargement contextuel de services, dispositifs contraints, gestion des dépendances.*

KEYWORDS: *service-oriented architecture, contextual services loading, constraint devices, dependency management.*

1. Introduction

Les environnements pervasifs peignent souvent des scénarii futuristes dans lesquels un utilisateur pourrait accèder à une panoplie de fonctionnalités, n'importe où, n'importe quand et à travers n'importe quel dispositif. Cependant, pour l'instant pareils scénarii éprouvent de grandes difficultés à se réaliser face à diverses problématiques, notamment la non adéquation des applications aux dispositifs mobiles. Nous soulevons ainsi le problème de la difficulté d'offrir aux usagers les fonctionnalités souhaitées face aux limites posées par les contraintes des ressources matérielles (CPU, mémoire, etc.) de leurs dispositifs mobiles.
Les travaux de [PAR 05], [GU 04] ou de [KIA 05] proposent des architectures conscientes du contexte, pour des environnements pervasifs, mais ommettent cependant d'adresser les difficultés concernant les plates-formes matérielles. D'autres travaux s'orientent plutôt vers les dispositifs contraints en ressources [GU 04] ou [REL 07]. Toutefois, ils ne traitent pas certains besoins tel que la conscience au contexte. Nous restons toutefois convaincus par la pertinence de la prise en compte du contexte [COU 05] afin de fournir une architecture qui soit adaptée aux environnements pervasifs et en même temps aux dispositifs mobiles.
Dans cet article, nous proposons une architecture de chargement contextualisé de services pour des dipositifs contraints. Nous basons notre approche sur trois éléments essentiels :

– Une architecture orientée-service : Le paradigme des services nous permet de facilement modéliser une fonctionnalités d'une application et nous offre les mécanismes de description, publication, découverte et de réutilisation.

– La prise en compte du contexte : nous définissons le contexte comme étant les données pouvant être recueillies à partir des services (nom, version, besoin en ressources, etc.), des dispositifs (capacité d'accueil, etc.), des utilisateurs (préférences).

– La technique du chargement/déchargement : elle permet de télécharger les entités nécessaires (dans notre cas les services) et de les décharger pour libérer les ressources matérielles en cas de besoin.

Charger un service implique de considérer les autres services dont dépend celui-ci. Nous évaluons en exploitant la description de ces services la possibilité ou pas de les accueillir sur le dispositif mobile. La décision du chargement d'un service et de ses dépendances se fait en tenant compte des besoins et des contraintes des plates-formes matérielles en terme d'offre de ressources (mémoire) et des préférences des utilisateurs (profil). Contextualiser le chargement des services optimise l'utilisation des ressources des dispositifs contraints, car ainsi ils n'ont pas à supporter la totalité des services et n'accueilleront dynamiquement que les services adaptés aux ressources dont ils disposent.
Cet article présente dans la section 2, un scénario illustrant le comportement souhaité de notre système, et dans la section 3, les travaux apparentés. Nous illustrons dans la section 4, le modèle de service considéré. Ensuite, l'architecture du système de

chargement de services est détaillé dans la section 5. Enfin, nous concluons dans la section 6, et donnons les perspectives de notre travail.

2. Scénario

Pierre est dans la salle d'attente de l'aéroport en train d'attendre l'embarquement. Il s'ennuie et décide d'écouter de la musique avec son téléphone portable. Il se rend compte qu'il ne possède pas le service nécessaire lui permettant d'écouter de la musique. Dans la même salle se trouve d'autres voyageurs possédant des dispositifs électroniques tels que des assistants personnels (PDA), des ordinateurs portables, des téléphones. Après découverte auprès de ces dispositifs et demande par Pierre, il installe un service de lecture de documents multimédia présent chez un de ses voisins. Celui-ci installe automatiquement l'ensemble des éléments composant ce service sur le PDA de Pierre, soit un service de lecture audio et un service de lecture vidéo. Après démarrage du service, Pierre peut écouter sa musique préférée, voir le clip vidéo correspondant à la chanson et voir la pochette du CD avec titre et durée des plages audio. L'attente se fait longue et Pierre voudrait maintenant jouer avec son téléphone. Or, ce dernier ne possède pas assez de ressource mémoire pour accueillir en même temps les services de lecture audio, d'affichage de clips vidéo et le jeu de Pierre. Ainsi, il décide de décharger une partie du service multimédia : le service de lecture vidéo vu qu'il n'en aura pas besoin en jouant, et accepte de charger le service du jeu en question. Il peut alors avec son téléphone jouer tout en écoutant de la musique.

3. Travaux apparentés

Notre état de l'art est orienté vers deux principaux axes. Le premier étant les architectures intergicielles pervasives, et le second relevant plutôt de l'ingénierie logicielle et portant sur les plates-formes de chargement abordés par la littérature. Nous nous intéressons précisemment à ce dernier point, car nous comptons optimiser le chargement de services.

3.1. *Architectures intergicielles pervasives*

Concierge [REL 07] est une implémentation minimale de la plate-forme OSGi [ALL 06], l'objectif de cette approche est de fournir une architecture d'exécution de services qui soit adaptée aux ressources limitées. Les auteurs relèvent le défi de fournir une telle plate-forme et prouvent sa performance sur une panoplie de dispositifs contraints. Cependant, ils ne proposent pas de solution pour un chargement optimisé des services qui viendront à y être exécutés. Dans la même perspective d'adaptation des architectures pervasives sur les machines réduites, [POL 04] propose une technique d'adaptabilité dynamique de services à l'exécution, de services dans un environnement pervasif. Un algorithme décisionnel est présenté, il prend en considération

plusieurs paramètres contextuels tels que les préfèrences utilisateurs, les contraintes materielles. Notre travail diffère du leur dans le sens où nous considérons le niveau service et non le niveau applicatif. Or, les services intègrent une notion de dépendance cruciale pour offrir à l'utilisateur la fonctionnalité souhaitée. [HOA 06] fournit un déploiement de composants, basé sur les contraintes dans les réseaux dynamiques. Les auteurs y considèrent une description formelle de composants à déployer, et adapte en fonction le déploiement, ils n'abordent cependant pas les contraintes en termes de ressources mémoires.

3.2. *Plates-formes de chargement*

Plusieurs mécanismes telles que la machine virtuelle java effectuent le chargement de fichiers de bytecode java [BRA 98]. Il s'agit d'un rapatriement statique dans lequel le nom de la classe à charger est fourni au chargeur de classes [ATT 03]. Le Class-Loader de Java ne permet qu'un simple rapatriement des fichiers et n'offre cependant pas de dynamicité du chargement, ni de conscience au contexte. D'autre part, d'autres ressources telles que les modules ou les composants peuvent être chargés. La plate-forme OSGi présente ce chargement décrit dans [HAL 04b] et [HAL 04a] comme palliant les limites du chargement classique des classes. D'autre part, [OBR 06] est un dépôt de bundles OSGi offrant un chargement statique des services, n'obéissant pas aux contraintes contextuelles. Des travaux tels que [ESC 05], sont venus améliorer l'architecture précédente et proposer ce même chargement de modules en abordant la problématique de résolution des dépendances entre différents bundles OSGi. [ESC 06], propose le chargement de ressources telles que les services OSGi mais en l'appliquant à la plate-forme .NET. Les services y sont encapsulés en assemblies (qui sont des modules .NET). Cette plate-forme reste limitée en terme de conscience au contexte et présente un autre problème qui est l'impossibilité de décharger des assemblies séparemment. Sun propose aussi une interface de chargement de services [MIC 06], il n'y ait toutefois pas question de conscience du contexte ni de dynamicité du chargement. Le tableau 1 présente un comparatif de quelques plates-formes de chargement de ressources.

Plateformes	Contexte	Dépendances	Chargement
`Class Loader`	non	non	fichier
`OSGi`	non	vérification locale au runtime	bundle
`OBR`	oui(serveur distant)	déploiement dynamique	bundle
`CLR .NET`	non	non	assemblie
`ServiceLoader`	non	non	service

Tableau 1. *Tableau comparatif de différentes plates-formes de chargement*

4. Modèle de service

Nous basons notre architecture sur les services que nous concevons suivant le modèle de [BRU 04]. Nous entendons par service, une unité logicielle capable d'être publiée, déployée et exécutée sur une plate-forme d'exécution. Un service est composé par un descripteur, des interfaces et le code applicatif (offrant la fonctionnalité). Un service offre des interfaces sur lesquelles des appels peuvent être faits par des tiers. Il est décrit par une méta donnée qui donne ses éventuelles dépendances. Les méta données descriptives d'un service seront plus détaillées dans la section 5.3.1. Nous considérons chaque service, ayant :

– des interfaces proposées : il s'agit des interfaces sur lesquelles pourront être réalisés les appels de méthodes venant de tiers.

– des interfaces requises : il s'agit des interfaces proposées par d'autres services et contenant les méthodes qui vont être utilisées par le service.

Les dépendances entre interfaces proposées et interfaces requises peuvent être satisfaites dans le cas où le service en question trouve à l'aide d'un protocole de découverte du contexte, un autre qui lui offre ce qu'il demande. Par ailleurs, elles peuvent être non satisfaites dans le cas où, par exemple, le service offrant l'interface souhaitée est absent. Le service A illustré dans figure 1 offre une interface, et requiert deux dépendances dont une est satisfaite par l'interface offerte du service B et une autre qui pourrait être satisfaite par le service C mais qui ne l'est pas, car celui-ci est absent.

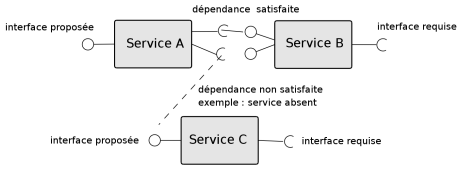

Figure 1. *Modèle de service considéré*

5. Chargement contextuel de services

En considérant le modèle de service précédemment décrit, nous construisons une architecture orientée-service, que nous détaillerons dans cette section.

5.1. *Architecture du système*

Notre approche prend en compte la description des services et des dispositifs, la publication et découverte de ceux-ci, la résolution des dépendances des services, la prise de décision quant à la possibilité de chargement ou pas de services, et le chargement même. La figure 2 illustre les différents éléments assurant ces fonctionnalités.

Afin de pouvoir découvrir dynamiquement les services se trouvant dans l'environne-

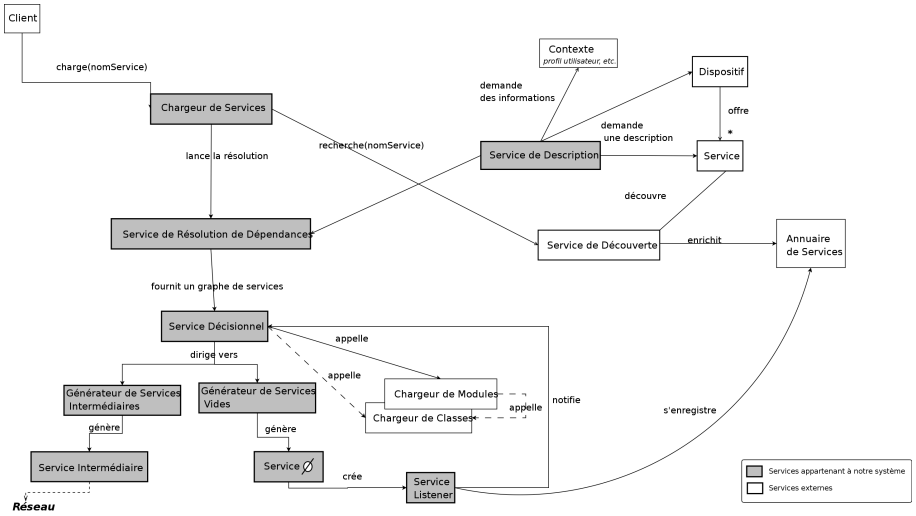

Figure 2. *Architecture du système de chargement de services*

ment, et d'adapter en fonction de cela le comportement de notre architecture, nous considérons la publication des services. Le *Service de Découverte* réalise la découverte (apparition et disparition de services), et enrichit l'*Annuaire*. Ce dernier recence la liste des services existant et découverts par le *Service de Découverte*. Outre la découverte, la prise en compte du contexte est assurée aussi à travers l'analyse de la description des services et des dispositifs qui est fournie par le *Service de Description*. Cette description peut être profilée pour prendre en compte le contexte (profil utilisateur, etc.). Au sein de ces descripteurs nous mentionnons les dépendances qu'ont les services avec d'autres. Lors du chargement, cette notion prend son importance car il est nécessaire de résoudre les dépendances des services à charger pour garantir à l'utilisateur une fonctionnalité. Le *Service de Résolution de Dépendances* se charge de la résolution des dépendances éventuelles d'un service avec d'autres.
Une fois tracé le graphe de dépendances des services est soumis à évaluation par le *Service Décisionnel* qui est capable de prendre des décisions sur le chargement éventuel des services. Il peut l'autoriser complètement, le refuser, ou décider du sous-ensemble de services dépendants à charger. L'issue de la décision peut aboutir soit vers le *Générateur de Services Vides* qui se charge alors de génèrer des services vides dans le cas où il est impossible de charger le service souhaité. [Un *Service Vide* étant un proxy qui retourne une valeur nulle à tous les appels de méthodes demandés, permettant d'éviter les interruptions brutales d'applications en cas de non résolution de dépendances]. soit, il aboutit vers un *Générateur de Services Intermédiaires* qui génère des services intermédiaires, sur le réseau par exemple, dans le cas où il y a impossibilité de chargement. [Un *Service Intermédiaire* est un proxy qui transforme les appels locaux en appel sur un service distant, par le réseau].

Enfin, le *Chargeur de Services* effectue le chargement des services en mémoire, ainsi qu'un certain nombre de services en dépendants. Au cas où certains services nécessiteraient le chargement de modules (composants, bundles, etc), il y a recours au *Chargeur de Modules*, ou bien au *Chargeur de Classes* pour charger les classes requises.

5.2. *Comportement dynamique du système*

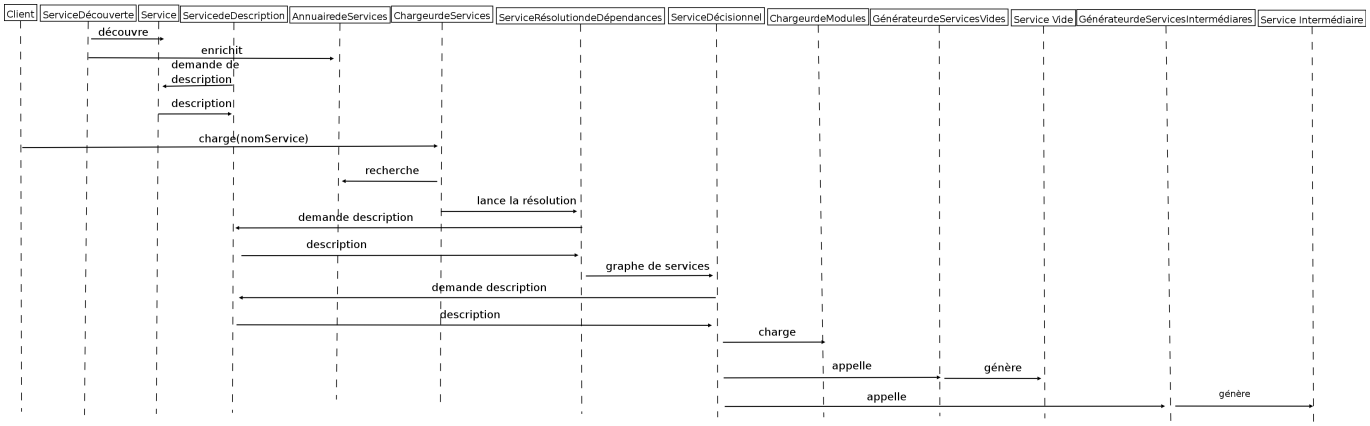

Figure 3. *Diagramme de séquence du système de chargement de services*

Un *Client*, illustré dans la figure 3, demande au *Chargeur de Services* de charger un service en précisant son nom. Ce dernier cherche le service en question dans l'*Annuaire*. Le trouvant il dirige un appel au *Service de Résolution de Dépendances* qui lance un appel vers le *Service de Description*, afin de dégager les dépendances du service à charger. Ce dernier intéragit avec le *Service Décisionnel*, en lui passant le graphe de dépendance des services. Le *Service Décisionnel* communique alors avec le *Service de Description* afin de récolter les données contextuelles des services et d'évaluer la possiblité de les charger et la capacité de la plate-forme hôte à les accueillir. Dans le cas où, il n'est pas possible de charger les services en question des appels vers *Générateur de Services Intermédiaires* ou le *Générateur de Services Vides* sont effectués pour tenter de remplacer les services manquants localement. La figure 4 présente le diagramme de séquence qui se déroule lorsque les services souhaités ne peuvent pas être chargés localement. Le *Service Décisionnel* évalue les données contextuelles relatives à la liste de services, obtenues du *Service de Description*. Dans l'impossibilité de charger certains services de cette liste en raison de leur non-conformité ou de leur grande taille par rapport à la plate-forme, etc. Le comportement du *Service Décisionnel* peut être de deux manières différentes :
- Il fait appel au *Générateur de Services Intermédiaires* qui génère un service intermédiaire qu'il aura trouvé sur le réseau par exemple et sur lequel nous pouvons effectuer des appels distants sans avoir recours au chargement.

- Il fait appel au *Générateur de Services Vides* qui génère un Service Vide qui répond à tous les appels de méthodes effectués sur lui. Celui-ci crée un *Listener* sur le vrai service, et l'enregistre auprès de l'*Annuaire* et le met à l'écoute de l'arrivée de nouveaux services satisfaisants sa requête. Dans ce cas, il notifie le *Service Décisionnel* afin d'évaluer la capacité de charger le nouveau service.

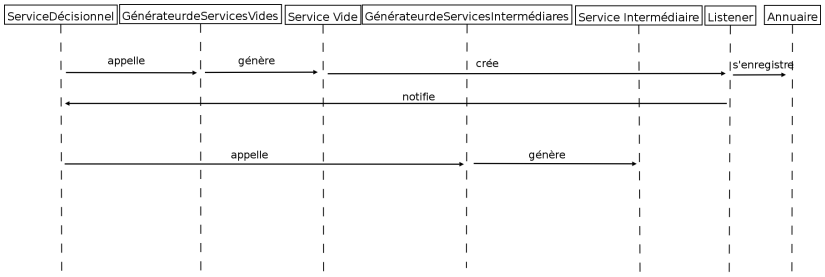

Figure 4. *Diagramme de Séquence alternatif*

5.3. *Description des principaux services*

5.3.1. *Service de Description*

– Description des dispositifs : Les plates-formes matérielles considérées peuvent être très restreintes en terme de ressources telles que la mémoire, le CPU, etc. Nous proposons d'adapter le chargement à ces périphériques de telle sorte que le chargement réalisé ne soit pas encombrant, et s'adapte aux configurations et disponibilités du dispositif accueillant les services. Dans notre système, nous ne considérons que la ressource mémoire. De ce fait, nous chargeons le *Service de Description* de collecter les données des périphériques utilisés. Le *Service Décisionnel* prend ces données et les évalue et les utilise comme référentiel afin d'autoriser ou pas le chargement. En effet, il sera impossible de charger des services de taille 100 Kilo Octets sur un téléphone ne disposant que de 50 Kilo Octets.

– Description des services : Le *Service de Description de Service* est chargé de la description contextuelle de chaque service. Les services possèdent des descriptions incluant le nom du service, sa description, sa version, mais aussi les liens requis par celui-ci, et les interfaces qu'il offre. Ces informations accompagnent généralement chaque service. Cependant, nous proposons d'enrichir nos services par des données contextuelles additionnelles. Ces données sont relatives à chaque contexte dans lequel le service est déployé. Elles sont récoltées à partir du contexte (profil utilisateur, etc.). Le *Service de Description* se charge de réunir les données nécessaires à la prise de décision relative au chargement des services. Outre les descripteurs propres à chaque service, nous enrichissons les services par des descriptions contextuelles. Ainsi, un service possède deux descriptions figure 5, une description statique et une autre contextuelle générée selon le contexte dans lequel évolue ce dernier. Le descrip-

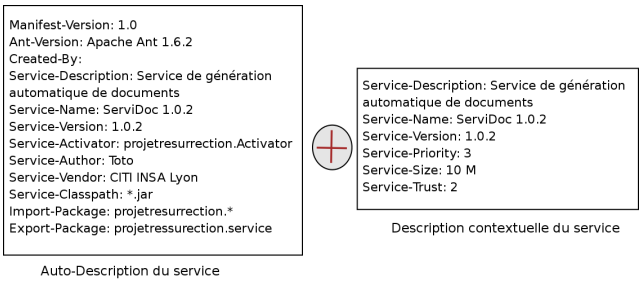

Figure 5. *Description d'un service*

teur du service inclut des données telles que les dépendances de celui-ci, le nom de son auteur, la version. Alors que le descripteur contextuel de service intègre :
- La priorité renseignant sur la necessité de charger ou pas le service. Dans le cas où il s'agit d'une haute priorité il est obligatoire de charger le service en question. Dans le cas contraire, il n'est pas utile de l'avoir sur le dispositif et nous gagnons ainsi l'espace de stockage qu'aurait pu occuper ce dernier.
- La taille informant sur l'occupation d'espace mémoire d'un service donné cette information est utile dans le sens où si le service est plus grand que l'espace disponible sur le dispositif, nous ne pourrions pas le charger et opterions dans ce cas à le remplacer par un autre consommant moins de mémoire, ou bien à faire un appel distant sur un autre service sans le charger sur le dispositif.
- La confiance qui est un degré attribué à chaque service, il traduit le niveau de sécurité accordé au service. Un niveau haut représente un risque élevé d'hébèrger un service malicieux.
Des éléments tels que la priorité ou la confiance pourront être attribués par rapport au contexte utilisateur, qui pourrait spécifier par défaut des critères sur ces services. Ainsi lors de la recherche d'un service la description rendue par le contexte est constitué de deux parties : la description statique du service et celle dynamique founie par le contexte.

5.3.2. *Service de Résolution des dépendances*

Le *Service de Résolution de Dépendances* est chargé de résoudre toutes les dépendances d'un service à charger. En se basant sur les descripteurs de chaque service il dégage la liste des services que celui-ci requiert et procède ainsi réccursivement jusqu'à obtenir un graphe de dépendance figure 6. Les graphes dégagés sont souvent de haute complexité en raison de la multitude des liens qu'ils incluent. Cependant, nous avons opté pour un graphe extrêment simplifié afin de retracer les dépendances pouvant être demandées par un service donné à charger. Charger un service reviendrait à charger l'ensemble du graphe qui en découle. Cependant, nous effectuons nos choix selon une stratégie de décision.

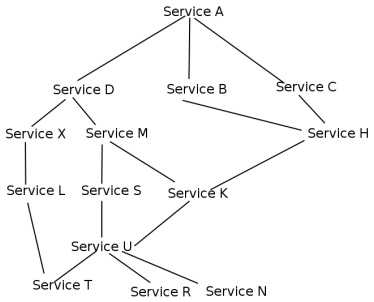

Figure 6. *Graphe de dépendances entre services*

5.3.3. *Service Décisionnel*

Lors du chargement contextuel nous tenons compte des données de la plate-forme accueillante, des préférences utilisateurs, et des données accompagnant les services mêmes. En effet, le chargement obéit à un processus de décision se basant sur les éléments auparavant listés. Pour cela, nous avons posé une politique de décision que nous illustrons à travers la figure 7. Supposons que l'utilisateur veuille charger le service

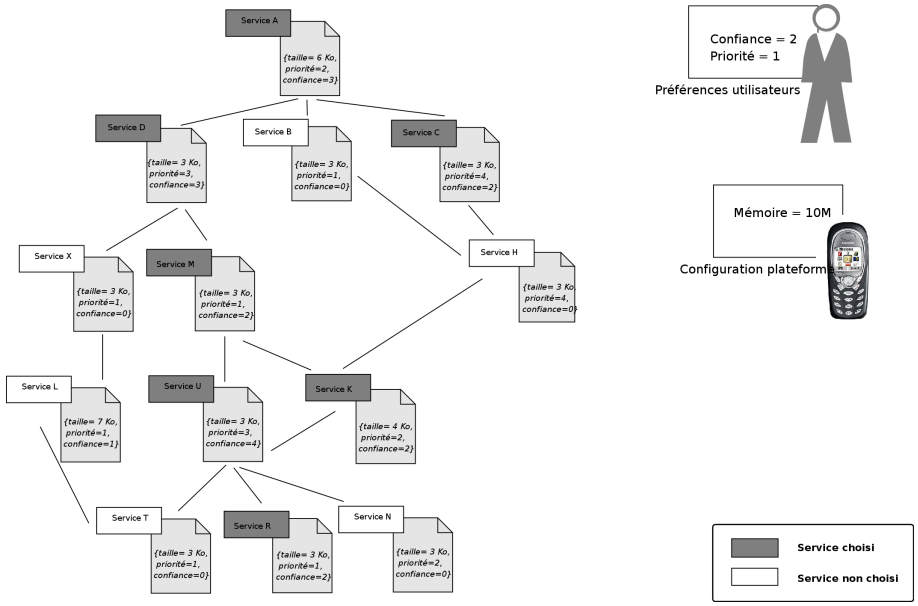

Figure 7. *Prise de décision lors du chargement*

A sur son téléphone. Charger le service A entraîne charger l'ensemble de ses dépendances illustrées par le graphe. Nous tiendrons compte des trois sources de données précédemment listées : l'utilisateur, la plate-forme et le service. L'utilisateur spécifie

dans ses préférences le niveau de confiance à partir duquel il accepte les chargements de services. Mais aussi, le seuil minimal de la priorité relative aux services à charger. D'autre part, le descripteur de la plate-forme matérielle illustre la mémoire restante que les services pourront occuper et qui n'excède pas 10 Méga Octets. Enfin, les descripteurs des services présentent la taille relative à chaque service, la priorité de ce dernier et son seuil de confiance. Dans le cas illustré, nous ne chargeons que les services ayant un niveau de confiance d'au moins 2, une priorité d'au moins 1 et pouvant être stockée dans l'espace disque restant. Notons que la taille mémoire devrait être suffisamment grande pour accueillir l'ensemble des services choisis à charger localement. Le processus de prise de décision est la résultante de l'application d'un algorithme de coloriage de graphes sur le graphe illustré. Cependant, il est necessaire d'effectuer une étude importante sur les paramètres à prendre en compte dans cet algorithme.

6. Conclusion et Perspectives

Nous avons proposé une architecture capable de charger contextuellement des services. Nous adoptons l'approche orientée-services pour concevoir notre plate-forme. Nous optimisons le chargement en tenant compte des données contextuelles de chaque service (taille, priorité), du profil utilisateur et du dispositif matériel et adaptons le chargement selon ces critères. En cas de non possibilité de chargement de toutes les dépendances de services, nous avons proposé un générateur de services intermédiaires locaux et un générateur de services vides. Ces générateurs permettent la continuité des applications même en cas de non résolution de dépendances. Le mécanisme de Listener permet de plus, lorsqu'un nouveau service est disponible ou plus de mémoire est libérée, de ré évaluer les dépendances et d'éventuellement remplacer les services vides. Nous réalisons notre plate-forme selon la spécification OSGi, sous felix et la technologie Java. A ce stade du développement nous réalisons l'extraction des dépendances d'un service dans un graphe, plus tard nous nous appliquerons à mettre en place le processus de décision. Des difficultés sont cependant rencontrées pour la gestion des dépendances entre services. Elle s'avère importante à cause de la complexité des liens pouvant exister. Une étude plus approfondie sur les stratégies de décision à adopter (algorithmes de coloriage est à réaliser. Le chargement de services ouvre des pistes à explorer et à inclure dans notre travail futur, en soulevant plusieurs problématiques :
- La disponibilité des services : A cause de la mobilité des utilisateurs, nous ne pouvons pas garantir la présence d'un service pendant une durée de temps précise. Une des techniques existantes pour pallier ce type de problèmes est le préchargement des données. Ainsi, jugeant qu'un service nous serait utile ultérieurement nous le préchargeons en mémoire, afin qu'il soit disponible en cas de besoin.
- L'encombrement de la mémoire : dans le cas de l'utilisation de dispositifs contraints la mémoire est une ressource précieuse. Afin de pallier l'encombrement de celle-ci nous envisageons de fournir un service de nettoyage ou *garbage collector*.
- Le chargement de services soulève la question de la sécurité. Charger un service malicieux sur sa propre machine est risqué. Nous avons pris en compte un critère de

confiance dans le descripteur de services, mais il serait impératif de considérer un référentiel commun, ou bien d'envisager une analyse préalable des services à charger.

7. Bibliographie

[ALL 06] ALLIANCE O., « OSGi- The Dynamic Module System for Java », http ://www.osgi.org/, 1999-2006.

[ATT 03] ATTSAT B., PANDA D., « Classloading in Oracle9iAS Containers for J2EE », rapport, 2003, Oracle corporation.

[BRA 98] BRACHA G., LIANG S., « Dynamic Class Loading in the Java Virtual Machine », *13th ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications*, 1998.

[BRU 04] BRUNETON E., COUPAYE T., LECLERCQ M., QUEMA2 V., STEFANI J., « An Open Component Model and Its Support in Java », *7th International Symposium on Component-Based Software Engineering CBSE7*, 2004, p. 7-22.

[COU 05] COUTAZ J., CROWLEY J., DOBSON S., GARLAN D., « Context is Key », *Commun. ACM*, vol. 48, 2005.

[ESC 05] ESCOFFIER C., DONSEZ D., « Implémentation de plate-formes dynamiques de services avec .NET », *RENPAR'16/CFSE'4/SympAAA'2005/Journées Composants*, 2005.

[ESC 06] ESCOFFIER C., DONSEZ D., HALL R. S., « Developping an OSGi-like Service Platform for.NET », *Consumer Cummunication and Networking Conference*, 2006.

[GU 04] GU T., PUNG H., ZHANG D., « A Middleware for Building Context-Aware Mobile Services », *IEEE Vehicular Technology Conference*, May 2004.

[HAL 04a] HALL R. S., « A Policy Driven Class Loader to Support Deployment in Extensible Frameworks », *Lecture Notes in Computer Science*, 2004.

[HAL 04b] HALL R. S., CERVANTES H., « an OSGi Implementation and Experience Report », *IEEE Consumer Communications and Networking Conference*, 2004.

[HOA 06] HOAREAU D., MAHÉO Y., « Constraint-Based Deployment of Distributed Components in a Dynamic Network », *Architecture of Computing Systems (ARCS 2006), LNCS, Frankfurt/Main, Germany,*, 2006.

[KIA 05] KIANI S. L., RIAZ M., LEE S., LEE Y.-K., « Context Awareness in Large Scale Ubiquitous Environments with a Service Oriented Distributed Middleware Approach », 2005.

[MIC 06] MICROSYSTEMS S., « Service Loader », http ://-java.sun.com/javase/6/docs/api/java/util/ServiceLoader.html, 2006.

[OBR 06] OBR, « Oscar Bundle Repository », http ://oscar-osgi.sourceforge.net/, 2006.

[PAR 05] PARK N.-S., LEE K., KIM H., « A Middleware for Supporting Context-Aware Services in Mobile and Ubiquitous Environment », *Proceedings of the IEEE ICMB*, Electronics and Telecommunications Research Institute, 2005.

[POL 04] POLADIAN V., SOUSA J., GARLAN D., SHAW M., « Dynamic configuration of resource-aware services », 2004.

[REL 07] RELLERMEYER J., ALONSO G., « Concierge : A Service Platform for Resource-Constrained Devices », *EuroSys Conference*, 2007.