



Rapport de magistère

Étude et réalisation d'une plate-forme domotique sur .Net

Soutenu par :
Clément ESCOFFIER

Sous la direction de :
Didier DONSEZ
Mikaël DESERTOT

Septembre 2004



Laboratoire LSR-IMAG
de Grenoble

Résumé

Ce rapport est consacré à l'étude et à la réalisation d'une plate-forme de services sur .Net. L'objectif du projet est de migrer une implémentation des spécifications OSGi existante écrite en Java (OSCAR) sur la nouvelle plate-forme de Microsoft. L'application résultante doit être compatible avec les spécifications d'OSGi.

Les principaux problèmes rencontrés concernent les contraintes liées aux plates-formes de services, ainsi que les possibilités de créer ou non un chargeur de classes sur .Net correspondant au Module Loader.

Pour cela, une étude détaillée du chargement de classes dans OSGi et des possibilités de charger et de décharger du code dynamiquement sur plate forme .Net a été réalisée.

Mots clés : Plate-forme de services, OSGi, Java, Chargement de classes, .Net, C#

Table des matières

Chapitre 1 : Introduction.....	3
1.1) Contexte institutionnel.....	3
1.1.1) Présentation du laboratoire LSR	3
1.1.2) Présentation de l'équipe	3
1.2) Présentation du projet.....	4
1.3) Plan.....	4
Chapitre 2 : Les plates formes de services.....	5
2.1) Architecture Orientée Service.....	5
2.2) Plate-Forme de Services.....	6
2.3) Objectifs et contraintes d'une plate-forme de services.....	6
2.4) Domaines d'applications.....	7
2.4.1) Passerelle domotique.....	7
2.4.2) Plate-forme véhiculaire.....	8
2.4.3) Plate-forme industrielle.....	8
2.4.4) Architecture à plugins.....	8
2.5) OSGi	9
2.5.1) Présentation	9
2.5.2) Atouts d' OSGi.....	10
2.6) Oscar	10
Chapitre 3 : Les enjeux d'un portage sur .Net.....	11
3.1) Pourquoi .Net ?	11
3.2) Objectif du portage.....	12
Chapitre 4 : Détails techniques du portage.....	14
4.1) Changement de langage de programmation.....	14
4.1.1) Présentation de J#.....	14
4.1.2) Présentation de C#	15
4.1.3) Microsoft Java Language Conversion Assistant.....	15
4.2) La problématique du chargement dynamique de classes	15
4.2.1) Chargement de classes en Java.....	15
4.2.2) Politique de chargement de classes d' OSGi	16
4.2.3) Présentation du système de chargement de .Net	17
Chapitre 5 : Travail effectué.....	20
5.1) Étude du chargement de classes sur Java et .Net	20
5.2) Module Loader	20
5.3) Tentative de création d'un chargeur d' assemblies	[Cervantes2004]
5.3.1) Chargeur d' assemblies entre Domaines d' Applications.....	21
5.3.2) Utilisation d'un seul domaine d'application.....	22
5.4) Tentative de court-circuitage du chargeur de .Net	23
5.5) Étude d'une implémentation d'une machine virtuelle .Net.....	23
5.5.1) Shared Source CLI.....	23
5.5.2) Architecture de .Net.....	24
5.5.3) Localisation et Chargement des assemblies sur Rotor.....	25
5.5.4) Déchargement de code.....	26
5.5.5) Assembly et chargement de classe.....	26
5.5.6) Modification du classloader des assemblies.....	27

Étude et réalisation d'une plate-forme domotique sur .Net

Chapitre 6 : Perspectives.....	28
6.1) Déchargement dynamique d' assemblies.....	28
6.2) Construction d'application dynamique sur .Net.....	28
6.3) Implémentation d'un plate-forme OSGi sur .Net.....	28
Chapitre 7 : Conclusion.....	29
Chapitre 8 : Autres documents de travail.....	30
Chapitre 9 : Bibliographie.....	31

Chapitre 1 : Introduction

1.1) Contexte institutionnel

1.1.1) Présentation du laboratoire *LSR*

Le laboratoire **Logiciels Systèmes Réseaux** (*LSR*) développe des méthodes et des outils spécifiques qui accompagnent la convergence des domaines du logiciel, des données et de la communication. En parallèle, les technologies du numérique connaissent une diffusion importante. Les formidables moyens de communication numérique qui se mettent en place sont à la fois une source d'études (protocoles, sécurité, sûreté, qualité de service, mobilité, ...) mais aussi le vecteur qui rend accessible partout et à tout moment de très grandes quantités de données et qui permet à des logiciels s'exécutant sur des processeurs distants (et éventuellement mobiles) d'échanger et de communiquer. Cette révolution technologique est la source de très nombreux problèmes scientifiques et techniques.

Le laboratoire est divisé en 7 équipes, chacune s'occupant d'un sujet précis :

- **ADELE** : Environnements pour le génie logiciel
- **DRAKKAR** : Réseaux et multimédia
- **PLIAGE** : Programmation logique et bio-informatique
- **SARDES** : Architecture et construction d'infrastructures logicielles réparties
- **SIGMA** : Systèmes d' Information : inGénierie et MultimédiA
- **STORM** : Services bases de données sur le réseau
- **VASCO** : Spécification, validation et tests de logiciels

1.1.2) Présentation de l'équipe

Les travaux de l'équipe Adele se sont toujours placés dans le contexte des logiciels industriels, c'est à dire des logiciels volumineux, très évolutifs et gérés par de nombreuses personnes.

Ses premiers travaux ont concerné la réalisation du gestionnaire de configuration Adele. Après des expérimentations réelles menées en environnement industriel, il en résulte que la difficulté principale rencontrée par les responsables d'entreprises et de projets est : la définition, le contrôle et le suivi des méthodes de développement qui permettent à la fois de garantir le respect des temps de cycle (de plus en plus courts) et de faire évoluer un logiciel complexe tout en garantissant sa qualité.

Par ailleurs les expérimentations ont mis en évidence le rôle crucial joué par la performance, la distribution et le facteur d'échelle en ce qui concerne les outils pour le génie logiciel industriel. Ces considérations ont un impact profond dans ses travaux. Pour être valide, une solution doit impérativement répondre à ces critères. Par exemple, le logiciel Adele, aujourd'hui distribué à plusieurs milliers d'exemplaires 1 tera Octets en ligne ainsi que les 1000 ingénieurs qui développent simultanément le même logiciel.

Ces dernières années ont vu une évolution très marquée de ses thèmes de recherches, qui sont passés de la gestion de configuration et des procédés aux fédérations et plate-formes à composants. Par contre n'ont changé ni le domaine (le génie logiciel) ni la méthode qui consiste à rechercher les problèmes réels, à construire des solutions réalistes et des prototypes de bonne qualité afin de les valider en exploitation réelle.

1.2) Présentation du projet

C'est dans ce contexte que je suis amené à effectuer mon projet de Magistère. Sous la direction de Mr. Didier Donsez, maître de conférence à l' Université Joseph Fourier de Grenoble, je m' intéresse à l'étude et à la réalisation d'une plate-forme de services sur l' environnement .Net. Pour cela, j'utilise une implémentation open sources d'une telle plate-forme conforme aux spécifications d' OSGi (Open Service Gateway Initiative - organisme proposant des spécifications d'une plate-forme de services), comme point de départ. Le but du projet est de porter cette application de Java vers la machine virtuelle .Net.

1.3) Plan

Ce rapport est organisé de la manière suivante Le chapitre 2 présente les principes des architectures orientés services, les spécifications OSGi et une implémentation de ces spécifications : OSCAR, point de départ du projet. Le chapitre 3 est consacré à .Net, la nouvelle technologie de Microsoft, ainsi qu'aux enjeux de porter les spécifications OSGi sur .Net. Le chapitre 4 donne un aperçu des problèmes rencontrés au cours de la migration d' OSCAR de Java vers .Net . Le chapitre 5 et 6 présentent le travail effectué et les perspectives possibles.

Chapitre 2 : Les plates formes de services

2.1) Architecture Orientée Service

Aujourd'hui, les logiciels « Change On Demand » sont devenus très populaires, les besoins changent vite et il faut s'adapter le plus rapidement possible. De nombreux producteurs de logiciels, proposent dorénavant cette solution évolutive. Une des approches pour réaliser ce genre de produit est une Architecture Orientée Services. Celle-ci est devenue très répandue avec l'explosion des Services Web.

Cette approche consiste à diviser le logiciel répondant à un problème, en un ensemble d'entités proposant des services. Chacune de ces entités peut utiliser les services proposés par d'autres entités. On obtient ainsi un réseau de services interagissant entre eux. Cette architecture s'appuie sur une architecture à composants (implémentation « réelle » des services [Cervantes2004]) et suit l'évolution logique des architectures logicielles [Endrei2004] (figure 1) :

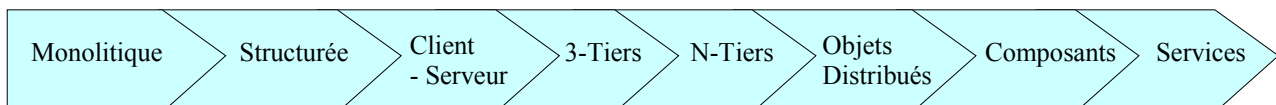


Figure 1 : Évolution des architectures logicielles

Les approches orientées services se caractérisent par :

- une transparence sur la localisation des services
- une indépendance des protocoles de communication
- une indépendance vis à vis des langages de programmation

L'infrastructure sous-jacente cache aux services ces détails. Il est aussi possible de substituer un service utilisé par un autre plus performant si celui-ci apparaît, ou par un service ayant un temps de réponse plus faible.

Une architecture orientée services est basée sur 3 acteurs principaux : l'annuaire de services, le fournisseur de services et le demandeur de services. La figure 2 illustre les interactions entre ces 3 acteurs.

Lorsqu'un service est activé, il s'enregistre auprès de l'annuaire (1). Ainsi, lorsqu'un autre service a besoin de lui, il peut le retrouver dans l'annuaire (2) et se lier à lui (3), pour ensuite pouvoir l'invoquer.

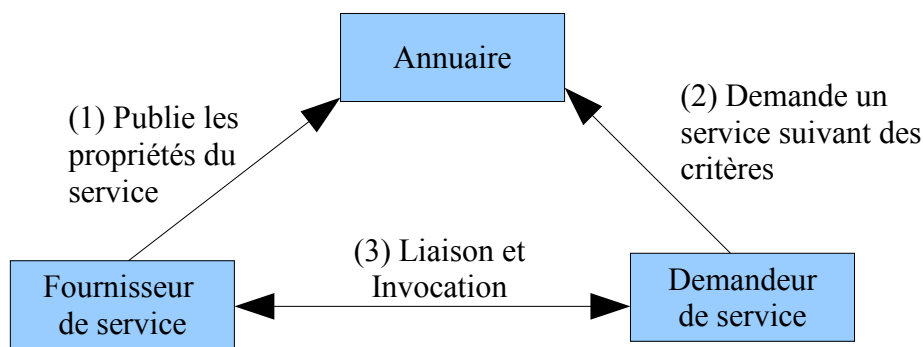


Figure 2 : Interactions entre les acteurs d'une architecture orientée services

Il existe déjà plusieurs technologies permettant de mettre en place l'architecture présentée dans la figure 2. Une des première fut Jini[Jini] proposé par Sun Microsystem basée sur Java. Mais il existe aussi OpenWings[OpenWings], Corba Trader, et les techniques utilisées dans les Services Web. Toutes ces solutions sont distribuées, c'est à dire que 2 services peuvent communiquer via le réseau. Mais il est parfois nécessaire de centraliser tous ces services sur une seule machine. Pour cela, on utilise des plate-formes de services permettant de mettre en oeuvre une approche orientée services de manière centralisée.

2.2) Plate-Forme de Services

Une plate-forme de services permet le déploiement de services, et gère la communication entre eux, mais cela de manière centralisée. Tous les services seront présents sur la même passerelle, ceci permettant une communication plus rapide entre eux. Mais cette passerelle est ouverte vers l'extérieur, et peut quand même communiquer via un réseau. Par exemple un service HTTP propose une interface qui permet par exemple d'administrer la plate-forme via un navigateur web. Un autre service peut par exemple offrir la possibilité de visionner les données livrées par des caméras connectées à la passerelle.

Un service peut en requérir d'autres. Par exemple le service visualisation peut fournir une interface web publiant les images des caméras. Ce service a besoin du service HTTP et du service camera pour fonctionner.

Une plate-forme de services doit permettre d'installer, de mettre à jour et de retirer ces services.

2.3) Objectifs et contraintes d'une plate-forme de services

L'objectif d'une plate-forme de services est de permettre le déploiement de systèmes basés sur la programmation orientée services. De plus, il ne doit pas y avoir d'interruption de services, c'est à dire que lorsqu' un service est mis à jour ou retiré, la plate-forme ne doit pas être arrêtée. En effet, des services « critiques » doivent pouvoir être exécutés comme des alarmes incendies ou des moniteurs cardiaques.

Il est donc nécessaire qu'une plate-forme de services gère de façon dynamique les services installés et leurs cycles de vie. Nous pouvons voir un exemple de cette gestion sur la figure suivante :

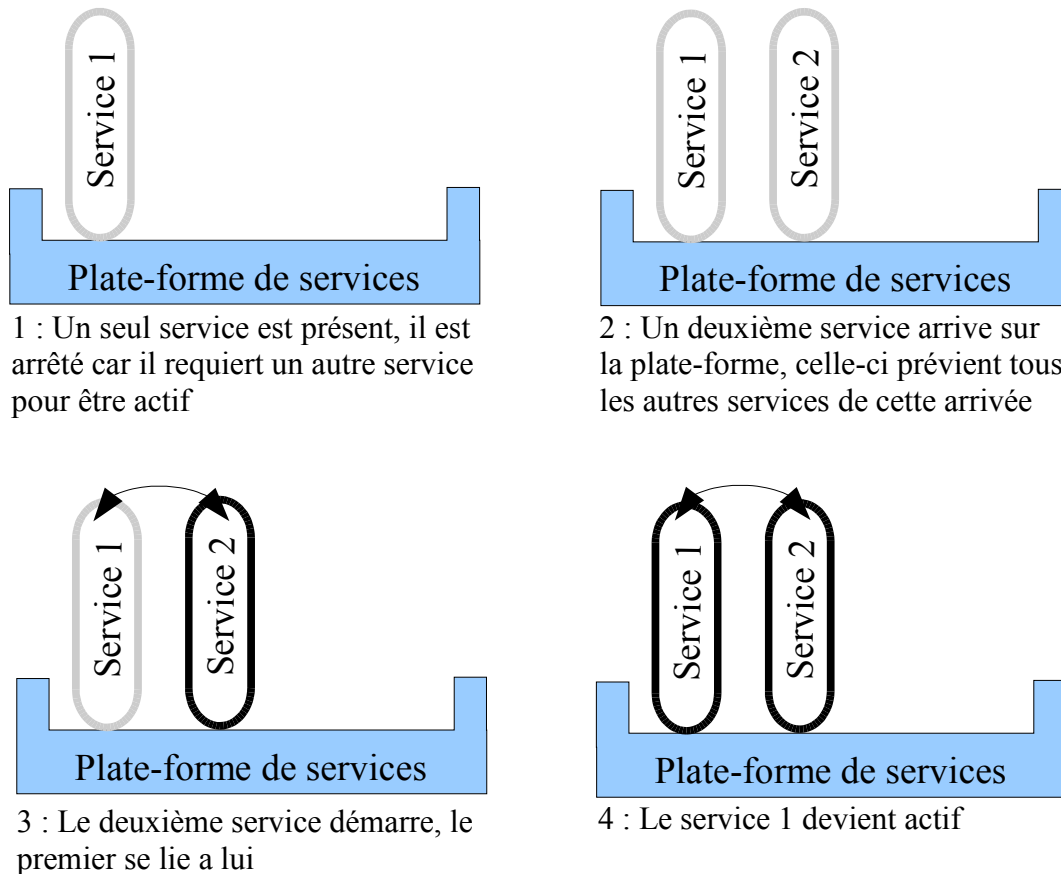


Figure 3 : Gestion dynamique des services

Une fois activé, le service peut être arrêté pour soit être mise à jour, soit être retiré de la plate-forme. Dans le cas d'une mise à jour, on revient dans le cas 2, c'est à dire que les 2 services sont stoppés. Si le service 2 est retiré alors on se retrouve dans le cas 1, le service 1 ne peut pas s'exécuter.

Toutes ces opérations ne doivent pas arrêter la plate-forme. Cette contrainte est résolue dans le cas de solutions distribuées de diverses façons. Par exemple, Jini utilise un système de bail renégociable. Les services utilisés, c'est à dire sans bail en cours, peuvent être arrêtés.

2.4) Domaines d'applications

2.4.1) Passerelle domotique

La domotique rassemble les technologies de l'informatique, de l'électronique et des télécommunications destinées au confort et à la sécurité de l'habitat. L'utilisation de plate-forme de services à l'intérieur du domicile permet son contrôle à distance. La figure 4 illustre ce concept.

Des prestataires de services installent, de façon dynamique, sans arrêt des autres services, des fonctionnalités sur la passerelle pour, par exemple, effectuer des diagnostics de panne des appareils

électroménagers, augmenter la sécurité du domicile (contrôle des caméras ...), ou médicaliser l'habitat (moniteur cardiaque ...).



Figure 4 :
Plate forme domotique

2.4.2) Plate-forme véhiculaire

L'utilisation de plate-forme de services dans le milieu de l'automobile est un secteur auquel s'intéressent plusieurs constructeurs. En effet, ceci permettrait de vendre au client des services autour de son véhicule comme l'appel d'urgence, l'aide à la navigation, la localisation de services (hôtels, pompes à essence, sites touristiques ...) La plate-forme peut aussi servir à diagnostiquer les pannes à distance.

2.4.3) Plate-forme industrielle

Le déploiement de plate-forme de services, pour surveiller et contrôler des capteurs et des machines, est une possibilité exploitée par l'entreprise Schneider Electric. Ceci permet de vérifier à distance le bon fonctionnement du réseau électrique par l'intermédiaire de capteurs reliés à la passerelle et d'agir en conséquence sur des actionneurs.

2.4.4) Architecture à plugins

Les plates-formes de services sont très intéressantes pour les architectures à plugins. C'est l'aspect dynamique proposé par ces plates-formes qui intéresse les développeur de logiciels à plugins. En effet, il est généralement très coûteux de redémarrer les applications utilisant ce type d'architecture, car il faut recharger tous les plugins. Par exemple l'environnement de développement Eclipse embarque une plate-forme de services afin de gérer ses plugins, permettant d'installer, d'arrêter et de désinstaller des plugins sans pour autant redémarrer l'application.

2.5) OSGi

2.5.1) Présentation

OSGi (Open Service Gateway Initiative) [OSGi03,Do2004,Kirk2002] est un consortium rassemblant plus de 75 compagnies et qui a pour objectif le développement de spécifications ouvertes pour la livraison de services. Parmi les compagnies appartenant au consortium, on retrouve IBM, Sun, EDF, France Telecom, Whirlpool, Schneider Electric, Nokia ...

La version 3 de ces spécifications est sortie en mars 2003. Il s'agit de spécifications pour une plate-forme de services répondant aux contraintes citées auparavant et adaptable a de nombreuses situations (figure 5), comme le montre la diversité des activités des entreprises supportant l'initiative.

OSGi est centré autour de Java et spécifie un framework (c'est à dire une plate-forme de travail) ciblant principalement les serveurs embarqués. Ce framework permet le déploiement des applications orientées services. Ce déploiement se fait par l'intermédiaire de « bundles » qui sont à la fois une unité de packaging (sous la forme d'un fichier Jar) et une unité de déploiement. Un bundle contient un ou plusieurs services. Il peut en proposer et en requérir d'autres. C'est le framework qui a en charge la gestion des dépendances entre services et la gestion de leur cycle de vie, c'est à dire qu'il doit contrôler l'installation, la mise à jour et le retrait de bundles de manière dynamique.

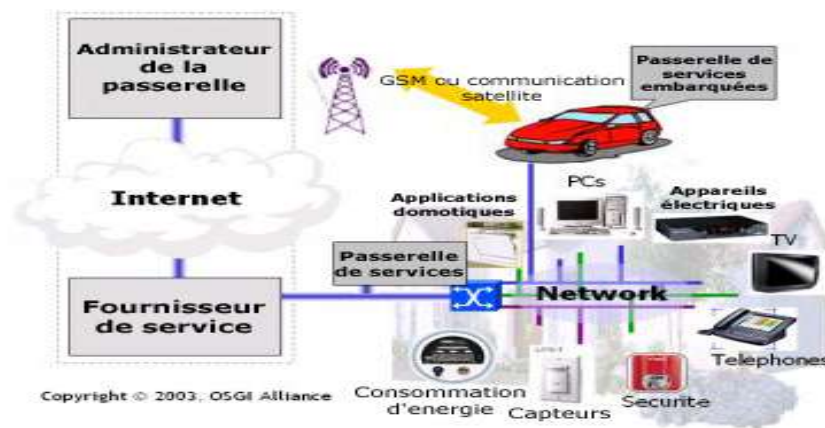


Figure 5 : Présentation de la plate forme OSGi

On peut représenter le framework OSGi (figure 6), comme une couche logicielle dans laquelle vient se brancher les bundles. Cette couche logicielle s'exécute sur la plate forme Java (Machine Virtuelle Java).

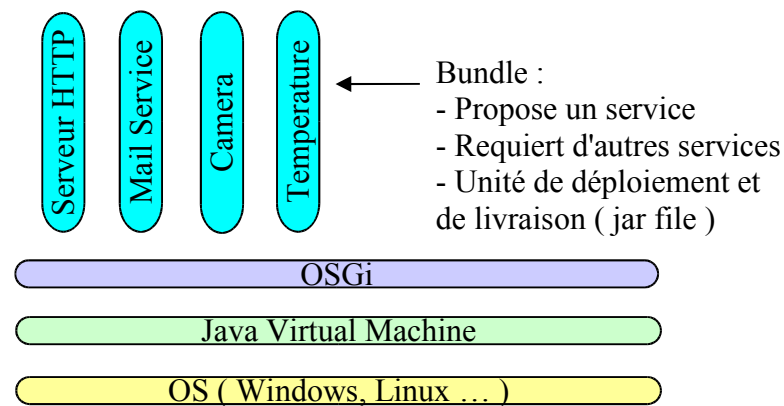


Figure 6 : Architecture de la plate forme OSGi

2.5.2) Atouts d' OSGi

OSGi tend à devenir un standard dans les plates-formes de services. La plate-forme spécifiée est adaptable pour de nombreuses situations et ne s'exécute pas forcément sur des serveurs embarqués. La gestion dynamique des bundles lui permet d'être utilisée pour toutes les applications à plugins, comme vu auparavant. C'est une plate-forme de ce type qui a été utilisée dans Eclipse pour permettre la gestion dynamique de plugins.

La spécification étant centrée autour de Java, il est possible d'installer une plate-forme OSGi sur n'importe quel système d'exploitation ayant la possibilité d'exécuter une machine virtuelle Java. Il existe aussi des implémentations de plate-formes OSGi basée sur Java Micro Edition (machine virtuelle Java destinée aux applications embarqués), ce qui permet d'installer OSGi sur des PC industriels, sur des PDA, voir même sur des téléphones portables.

2.6) Oscar

Oscar [Oscar, Hall2004] est une implémentation open source des spécifications d' OSGi. Elle a été réalisée par le Dr Richard HALL, appartenant à l'équipe ADELE du laboratoire LSR. OSCAR fait parti du consortium ObjectWeb. Cette implémentation respecte la majeure partie de la version 3 des spécifications. Elle a servi de point de départ au projet.

Chapitre 3 : Les enjeux d'un portage sur .Net

Le but du projet est de porter OSCAR sur .Net. Ceci permettrait d'avoir une plate-forme de services fonctionnant sur une autre machine virtuelle que Java, et proposant les mêmes fonctionnalités.

3.1) Pourquoi .Net ?

.Net [Thai2003] est la solution que Microsoft a créée pour concurrencer Java. L'implémentation principale est la plate-forme .Net proposée par Microsoft mais cette implémentation n'est disponible que sous Windows. Il existe d'autres implémentations de .Net comme Mono qui est une version libre et développée avec le soutien de Ximian. Cette implémentation fonctionne sous Windows mais aussi sous Linux. Microsoft a aussi partagé une partie des sources de sa plate-forme .Net sous la licence Shared Sources. Les Shared Sources CLI permettent d'avoir une machine virtuelle .Net très proche de celle de Microsoft qui fonctionne à la fois sous Windows, sous BSD et depuis peu sous Linux.

La technologie .Net propose une machine virtuelle relativement proche de Java. De plus, elle a la possibilité de charger et de décharger du code dynamiquement, propriété indispensable pour implémenter les spécifications OSGi. En effet, c'est cette capacité qui permet de ne pas arrêter la plate-forme lors de l'ajout ou du retrait de services.

.Net et Java ont plusieurs points communs. Leur unités de packaging se ressemblent beaucoup (les fichiers Jar pour Java, les assemblies [MSDN01] pour .Net). Ce sont 2 conteneurs physiques de classes qui ont les mêmes éléments de bases comme l'on peut voir sur le schéma 7.

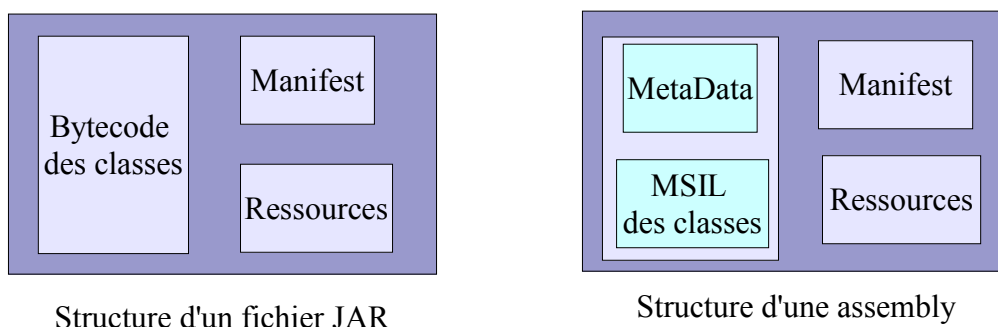


Figure 7 : Comparatif des conteneurs de classes sur les 2 plates formes

Les fichiers Manifest, donnant les informations principales sur le fichier Jar ou sur l'assembly, n'ont pas la même syntaxe mais ont la même fonction. Le Manifest d'une assembly est un fichier XML alors que celui d'un fichier Jar est un fichier texte. .Net sépare le code des classes (MSIL), des méta-données, cette séparation n'existant pas dans Java.

Les 2 machines interprètent du code portable (c'est à dire ne dépendant pas du système d'exploitation, ni de l'architecture de l'ordinateur) ressemblant à un assembleur (Bytecode pour

Java, MSIL (Microsoft Intermediate Language) pour .Net). De plus, elles utilisent une compilation Just In Time (Juste à Temps). Cette technique consiste à transformer le code des classes (MSIL ou Bytecode) en code exécutable juste avant l'exécution. Ceci permet de ne compiler que les classes nécessaires.

Java possède un répertoire spécial dans lequel il partage toutes les classes contenues. Ce répertoire est généralement /lib/ext du répertoire contenant le Java Runtime Environnement (JRE). .Net possède aussi un répertoire de ce type nommé Global Assembly Cache, dans lequel la plate-forme stocke toutes les assemblies utilisées.

Mais .Net apporte aussi des avantages par rapport à Java. Sur la plate-forme Java, il n'existe qu'un seul langage de programmation : Java. Sur .Net, il existe plusieurs langages, les plus connus sont J#, C#, et VB.net. Mais il est possible d'écrire des programme en C++ (C++ managé), en Python (avec Iron Python), en Perl (PERL.Net) mais aussi en Cobol.... Cela laisse au programmeur le choix du langage de programmation sans se soucier si son travail sera compatible avec le reste de l'application. Il est aussi possible de créer son propre langage pour .Net. Cette fonctionnalité est possible grâce à la Common Langage Specification (CLS). La CLS est en fait la spécification commune à tous les langages. Cette CLS n'autorise que les langages objets, et exclue donc les langages procéduraux tel que le langage C.

.Net propose 2 types d' assemblies : les assemblies avec un nom fort et les assemblies avec un nom simple. Les assemblies avec un nom fort (ou strong name assemblies) possèdent plus d'information que les autres assemblies. Ces informations permettent à la machine virtuelle d'identifier exactement une assembly et de choisir la « meilleure » si plusieurs assemblies conviennent. Les assemblies avec un nom simple n'ont qu'un nom, alors que celles avec un nom fort doivent spécifier :

- un nom
- un numéro de version
- une culture / langage
- une clé public (clé de hachage de l'assembly)

Le fait d'avoir tous ces renseignements est très utile pour la vérification des données chargées, et pour identifier exactement quelle assembly doit être chargée. Dans Java, seul le nom complet de la classe (c'est à dire le nom du package suivi du nom de la classe) permet d'identifier une classe.

Un autre avantage de .Net, est sa capacité à pouvoir utiliser en même temps 2 versions d'une même assembly si les 2 applications les utilisant ne communiquent pas grâce à la notion de domaines d'applications.

De plus, Microsoft propose un environnement d'exécution .Net allégé, afin d'être utilisé dans les applications mobiles. Cette implémentation, nommée Compact .Net, est utilisée dans les Pocket PC ou dans les téléphones portables. Cette plate-forme peut être considérée comme l'équivalent .Net de Java Micro Edition.

3.2) Objectif du portage

Le but du portage est de savoir s'il est possible de concevoir une plate-forme de services compatible avec les spécifications OSGi s'exécutant sur la machine virtuelle .Net.

Un des objectifs du projet est que le framework OSGi.net (figure 8) ne dépende pas de

Étude et réalisation d'une plate-forme domotique sur .Net

l'implémentation de la plate-forme .Net sur laquelle il s' exécute. De plus nous voudrions utiliser les avantages qu'offre .Net pour pouvoir supporter des bundles écrits dans plusieurs langages (bien que cette particularité ne soit pas encore supportée pour toutes les implémentations de .Net), ainsi qu'exécuter 2 versions d'une même assembly en même temps.

Nous désirons donc savoir si le résultat suivant est envisageable :

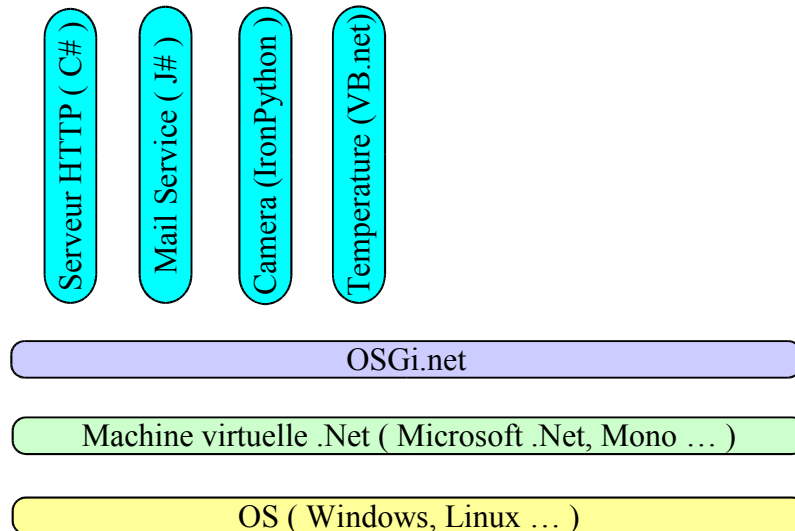


Figure 8 : Architecture désirée du projet

OSGI.net est le framework OSGi s'exécutant sur la plate-forme .Net. Le fait que cette plate-forme propose plusieurs langages permet de créer des bundles hétérogènes, sans se soucier de la communication entre eux.

Si cette architecture est possible, alors le but du projet est de l'implémenter en partant d'Oscar.

Chapitre 4 : Détails techniques du portage

Le portage d'une application Java à une application .Net pose plusieurs problèmes. Le premier problème est qu'il ne s'agit pas du même langage de programmation. Il y a des différences de syntaxe. Le deuxième problème technique est le changement de machine virtuelle. En effet, même si les structures des 2 machines virtuelles se ressemblent beaucoup, il existe des différences. Nous aborderons ultérieurement le problème du chargement de classes, principal problème rencontré.

4.1) Changement de langage de programmation

Nous avons vu que la plate-forme .Net supporte plusieurs langages. Nous avons considéré les 2 principaux langages de .Net pour le portage, c'est à dire J# et C#. Les autres langages ne correspondent pas à nos attentes. En effet, VB.Net est plus destiné aux applications visuelles, PERL.Net et Python sont des langages interprétés et seront trop lents à l'exécution par rapport à J# et à C#.

4.1.1) Présentation de J#

Une des principales caractéristiques techniques du langage J# est que c'est une implémentation du langage Java. En effet, Microsoft a pris comme base pour le J#, la version 1.1.4 du JDK (Java Development Kit) de Sun et l'a modifié pour créer le J#, afin d'en faire un langage compatible avec le framework .Net et concurrençant Java.

De ce fait, le J# intègre:

- un support des classes Java de la version 1.1.4 excepté deux grandes implémentations, le RMI (Remote Method Invocation) et le JNI (Java Native Interface) qui permettent respectivement d'exécuter du code sur un serveur ainsi que de récupérer le résultat, et d'utiliser du code non Java (C ou C++ par exemple). Cependant la version 1.1.4 n'est même pas compatible avec le Java 2 (qui commence à partir de la version 1.2) et donc avec la dernière version de Java (actuellement 1.4.2).
- un support du COM (Component Object Model) comme son prédécesseur, le J++.
- un support natif des services Web XML grâce au framework .Net.
- une inter-opérabilité totale avec les autres langages orientés .Net (C#, VB.Net,...).

De plus, le J# est fait de telle sorte que le Java puisse cohabiter avec des applications .Net cependant il ne s'agit logiquement que de la version 1.1.4 de Java. J# a été créé pour permettre l'utilisation d'application Java déjà existante sur la plate-forme .Net.

Pour utiliser J#, il faut compiler les sources d'Oscar avec le compilateur J#. Malheureusement, Oscar est écrit en Java 1.4, et les différences entre les 2 versions de Java sont importantes. Oscar utilise des classes qui ne sont pas disponibles dans J#. Donc l'utilisation du langage J# n'est pas possible.

4.1.2) Présentation de C#

En 1996, Microsoft a embauché Anders Hejlsberg, travaillant à l'époque pour Borland sur Delphi. Il avait pour objectif la mise au point d'un système pour rendre le développement plus aisé et plus rapide. Il s'en suivit l'architecture .Net et le langage C# [Drayton2002, Thai2003] qui devient aussitôt la référence et le principal langage pour Microsoft.

Le C# est dérivé du C++, on y retrouve aussi plusieurs caractéristiques des langages relativement récent comme Java. Puisque le C# s'inscrit dans la lignée C / C++, il en reprend les principales caractéristiques mais y ajoute des caractéristiques de Java :

- Orienté objet : tout doit être incorporé dans les classes.
- Garbage collector (suppression des objets inutilisés automatiques)
- Disparition des pointeurs
- Manipulations des tableaux comme en Java
- Passage de tableaux en arguments
- Disparition de l'héritage multiple de C++, mais possibilité d'implémenter plusieurs interfaces dans une classe.
- Chargement / Déchargement de code dynamique possible

Le langage C# paraît correspondre à nos attentes pour le portage. De plus Microsoft fournit un outil pour transformer du code Java en C#.

4.1.3) Microsoft Java Language Conversion Assistant

Microsoft propose un outil pour transformer une application Java existante en C# : Microsoft Java Language Conversion Assistant. Cet outil est à l'heure actuelle dans sa version 2. Il permet de migrer facilement des applications existantes écrites en Java vers la plate-forme .Net. La migration n'est pas parfaite, et il est clairement stipulé que les applications utilisant des chargeurs de classes personnalisés ne sont pas migrables automatiquement. De plus la notion de domaine d'application n'existant pas en Java, le programme résultant de la migration n'utilise pas cette notion.

4.2) La problématique du chargement dynamique de classes

Il existe des différences entre les 2 machines virtuelles. Elles ne proposent pas les mêmes possibilités techniques. Une des différences fondamentales concerne le système de chargement de classes.

De plus la migration du reste de la plate-forme, ne pose pas de problèmes techniques et a déjà été réalisée à l'Université Libre de Berlin [Ntuba2004].

4.2.1) Chargement de classes en Java

Le chargement dynamique de classes [Halloway2001,Neward2001,Neward2001a] est une possibilité très importante que propose la machine virtuelle Java. Ce mécanisme permet d'installer

des composants logiciels (de nouvelles capacités) à une application en cours d'exécution. Une des premières utilisations de cette technologie concerne les applets java : le navigateur télécharge les classes java de l' applet, les lance sans redémarrer le navigateur.

D'autres langages avant Java possédaient cette possibilité comme Lisp et Pascal, mais seul Java proposait un chargement « paresseux » (c'est à dire ne chargeant qu'au moment de l'exécution de la classe (Just In Time)), sûr d'un point de vue du typage, personnalisable via les objets classloader (chargeur de classes), et multi-localisation (chaque classloader peut charger des classes depuis un endroit différent) [Vernners1997].

Par défaut, la machine virtuelle Java contient 3 chargeurs de classes :

- bootstrap classloader : chargeant les classes de base de Java
- extension classloader : chargeant les classes contenues dans le répertoire extension de Java
- classpath classloader : chargeant les classes du classpath (indiqué par l'utilisateur)

Si un des chargeurs ne trouve pas la classe demandée, il y a un mécanisme de délégation. Java gère une hiérarchie de classloader. Lorsqu'un des classloaders échoue alors il demande à son « père » de rechercher la classe, et ceci jusqu' au bootstrap classloader (qui n'a pas de père). Si le bootstrap classloader n'arrive pas a charger la classe alors une exception est levée : le chargement a échoué.

Java permet la création de nouveaux objets classloader pouvant charger des classes depuis une localisation personnalisable.

Le déchargement des classes est fait par le Garbage Collector, sur le classloader. Lorsque le chargeur n'est plus utilisé, les références vers lui sont supprimées puis le classloader est également supprimé à son tour. Ensuite les classes chargées par le classloader sont supprimées de la mémoire vu que leurs références vers leur classloader n'existent plus.

4.2.2) Politique de chargement de classes d' OSGi

Les spécifications OSGi [Hall2004a] imposent un système de chargement de classes différent de celui de Java. Ceci est tout de même possible car Java permet la redéfinition des chargeurs de classes. Chaque bundle possède son chargeur de classes. La politique de chargement est la suivante (voir figure 9):

- Si une classe n'est pas chargée (C1) et qu'elle se situe dans le bundle (bundle 2), alors la classe est chargée
- Si une classe n'est pas chargée et qu'elle ne se situe pas dans le bundle (C2), alors la politique de chargement de classes de Java est utilisée. Le chargeur va chercher la classe de la machine virtuelle.
- Si une classe est déjà chargée dans un autre bundle (C3), alors c'est la classe déjà chargée qui est utilisée. Pour cela le chargeur de classes du bundle 1 passe la classe au chargeur de classes du bundle 2.

Ceci implique que lorsqu'une classe est déjà chargée par un bundle, tous les autres bundles doivent

l'utiliser. Ce système permet d' éviter les conflits de version entre les différentes implémentations des classes.

Les spécifications d' OSGi précisent que certaines classes peuvent ne pas être exportées. C'est à dire qu'il est tout de même possible de créer des classes « privées » qui ne pourront pas être utilisées par d' autres bundles et des classes « publiques » exportables, utilisables par les autres bundles.

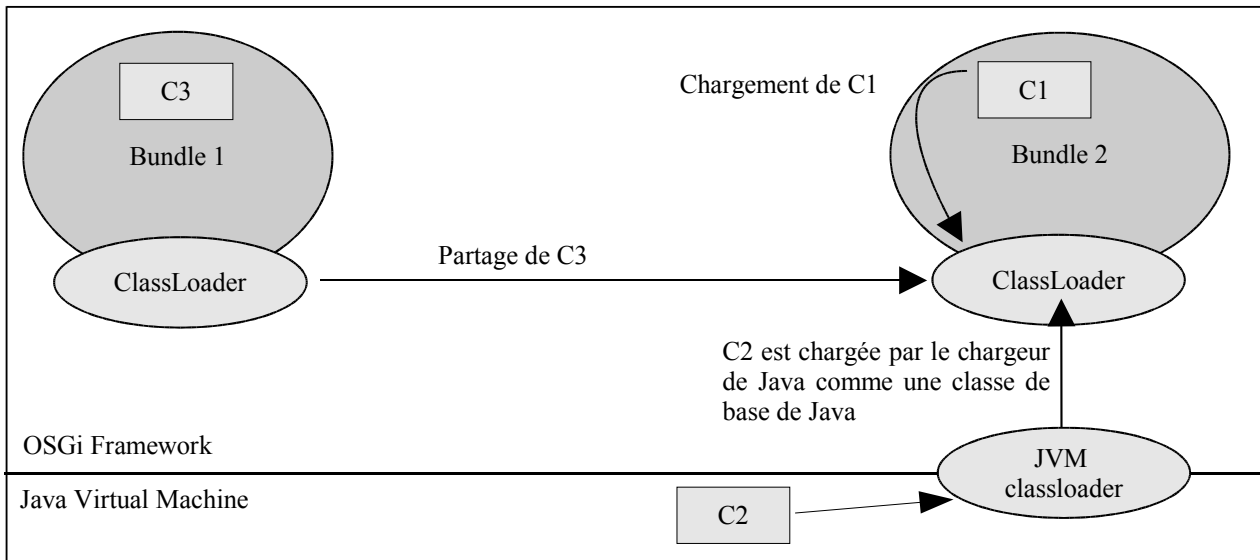


Figure 9 : Chargement de classes dans OSGi

4.2.3) Présentation du système de chargement de .Net

Dans la plate-forme .Net, les classes se trouvent obligatoirement dans des assemblies. Auparavant, nous avons vu qu'une assembly était un conteneur physique de classes. Toutes les classes qui sont utilisées par la plate-forme doivent se trouver dans un de ces conteneurs. C'est une différence avec Java, dans lequel l'utilisation de classes individuelles (c'est à dire sous la forme d'un fichier .class) est possible.

Lors du chargement, la plate-forme ne charge pas les classes une par une, mais charge une assembly en entier [MSDN02]. Une fois chargée, toutes les classes de l' assembly sont disponibles. Mais elles ne sont pas disponibles partout. En effet, .Net apporte une nouvelle notion dans l'environnement d'exécution : le Domaine d' Application (figure 10). Un Domaine d' Application est un environnement d' exécution isolé. Plusieurs domaines d' applications peuvent être exécutés au sein d'un même processus. Ceci permet d' avoir un seul processus pour la plate-forme .Net, exécutant plusieurs domaines d'applications. Généralement, il y a un domaine d'application par application. Toutes applications .Net s'exécutent dans un domaine d'application. Mais ces domaines doivent utiliser des techniques de communication inter-processus pour communiquer entre eux.

Les assemblies chargées à l' intérieur d'un domaine, ne sont pas disponibles dans les autres domaines. L' assembly Asm1 n'est pas disponible dans le domaine d'application AppDomain 3. Par défaut, chaque application .Net s'exécute dans un domaine portant le nom de l'application. Mais une

application peut créer d'autres domaines. La communication entre ces domaines n'est possible qu'en utilisant la technologie .Net Remoting, équivalent de Java RMI sur .Net, ceci affectant les performances de l'application. En effet, pour passer un objet d'un domaine d'application à un autre, il faut sérialiser l'objet, puis le passer à la couche réseau qui transmettra l'objet sérialisé, ensuite le deuxième domaine doit dessérialiser l'objet pour le rendre de nouveau utilisable. Ces indirectes prennent du temps alors que les 2 domaines d'application sont sur la même machine.

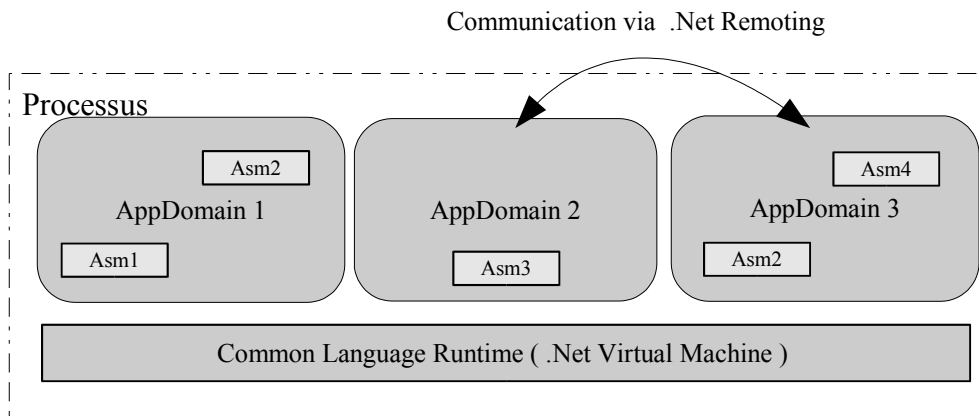


Figure 10 : Les Domaines d' Applications sur .Net

Chaque domaine d'application gère ses assemblies, c'est à dire le chargement de celle-ci. Il ne les décharge que lorsque le Domaine est détruit.

Le principe de chargement sur la plate-forme .Net est le suivant :

- Si une classe est déjà dans un assembly chargé dans le domaine, elle est forcément réutilisée
- Si ce n'est pas le cas, alors l' environnement d'exécution recherche l' assembly conteneur puis la charge, la classe contenue est ensuite utilisée.

Une autre différence entre .Net et Java concerne les localisations, où l'environnement d'exécution recherche les assemblies / classes. En Java, les classes chargeables sont :

- Soit dans le « classpath »
- Soit dans le répertoire extension (/lib/ext)
- Soit dans le JRE (c'est à dire les classes de base de Java)
- Soit dans une localisation spécifiée par un chargeur de classe personnalisé

Alors que sur .Net, l'environnement d'exécution recherche dans les endroits suivant :

- Le Global Assembly Cache
- Des emplacements physiques spécifiés dans les fichiers de configuration de l'application de manière statique [MSDN03], en non modifiable en cours d'exécution

- Demande à l'utilisateur l'emplacement via Microsoft Windows Installer

Les fichiers de configurations permettant de spécifier les emplacements des assemblies sont des fichiers XML. Il est possible de configurer les localisations des assemblies ainsi que les versions utilisées. Il existe 3 différents fichiers, se rapprochant de la politique de chargement de Java :

- l' *application configuration file*, équivalent au classpath et aux localisations des chargeurs de classes personnalisés
- le *publisher configuration file* pour les assemblies situées dans le Global Assembly Cache, équivalent au répertoire extension de Java
- le *machine configuration file* pour toutes les applications de la plate-forme .Net, pour les classes de base de .Net

Chapitre 5 : Travail effectué

Le travail que j'ai effectué jusqu'à maintenant a surtout été une étude sur le fonctionnement de .Net, et de ses possibilités par rapport à Java. J'ai aussi testé plusieurs systèmes de chargement dynamique d'assemblies afin de construire un chargeur d'assemblies respectant les spécifications d'OSGi.

5.1) Étude du chargement de classes sur Java et .Net

Après avoir découvert OSGi et OSCAR, la majeure partie de mon travail a concerné l'étude du système de chargement de classes sur Java et sur .Net. Le résultat de cette recherche a été l'écriture d'un document concernant le chargement dynamique sur .Net. (voir autres documents de travail)

5.2) Module Loader

OSCAR utilise un chargeur de classe spécial appelé Module Loader crée par Dr Richard HALL. Ce chargeur est une solution générique, et adaptable aux problèmes de chargement des applications Java comme les plates-formes à composants, les architectures à plugins ... La principale faiblesse des chargeurs de classes Java est qu'ils sont statiques et qu'il n'est pas possible de modifier les localisations explorées. De plus Java ne gère pas les différentes versions d'une classe.

Pour résoudre ces problèmes, le Module Loader utilise la notion de Module. Un Module est un ensemble de classes et de ressources. Il possède son propre chargeur de classes permettant de charger une ressource à partir du module. Les modules sont gérés par un Module Manager. Ce dernier gère la politique de recherche des ressources. Le principal avantage du Module Loader est la possibilité de créer et de détruire dynamiquement de nouveaux modules. Le Module Manager communique avec les modules grâce à leur chargeur de classes. Le fait que chaque module aie son propre chargeur de classes permet de charger différentes versions d'une même classe dans différents modules. En effet en Java, une classe chargée est identifiée par son nom et par son chargeur de classes.

Cette solution est adaptable à tous les problèmes de chargement de classes en Java, car la politique de recherche et de chargement est implémentée dans le Module Manager. C'est lui qui va gérer le partage d'une ressource entre 2 modules. Trois politiques de chargement sont exposées par le Module Loader, mais il est possible d'implémenter d'autres politiques :

- Recherche intra-module : les ressources chargées se trouvent forcément dans le module
- Recherche globale : les ressources peuvent être chargées à partir de n'importe quel module
- Politique d'import / export : chaque module spécifie les ressources exportées, celles-ci sont utilisables par les autres modules

Dans OSCAR, chaque bundle est un module, le Module Manager se trouve dans le framework. Afin de répondre aux spécifications OSGi, c'est la politique d'import / export qui est utilisée.

5.3) Tentative de création d'un chargeur d'assemblies

Après avoir étudié les techniques de chargement de la plate-forme .Net, j'ai implémenté 2 différents chargeurs d'assemblies devant correspondre au Module Loader. Malheureusement, aucun des 2 chargeurs ne répond correctement aux contraintes, mais ils permettent de mieux comprendre le fonctionnement de la plate-forme .Net.

5.3.1) Chargeur d'assemblies entre Domaines d'Applications

La première tentative consistait à considérer l'équivalence entre un bundle et un domaine d'application (appDomain) . Un domaine d'application spécial (AssemblyLoader) a été implémenté. Il est au centre de ce test.

Cette solution se rapproche beaucoup du Module Loader. On peut considérer l'AssemblyLoader comme le Module Manager et chaque AppDomain comme un Module. La politique de recherche et de chargement de classes est implémentée dans l'AssemblyLoader, il est possible de créer de nouveaux modules / AppDomain de façon dynamique ainsi que de les détruire.

Cette approche apportait plusieurs avantages :

- un bundle / domaine d'application possède son propre chargeur comme dans le Module Loader
- il est possible de faire cohabiter plusieurs versions d'une même classe sur la plate-forme OSGi

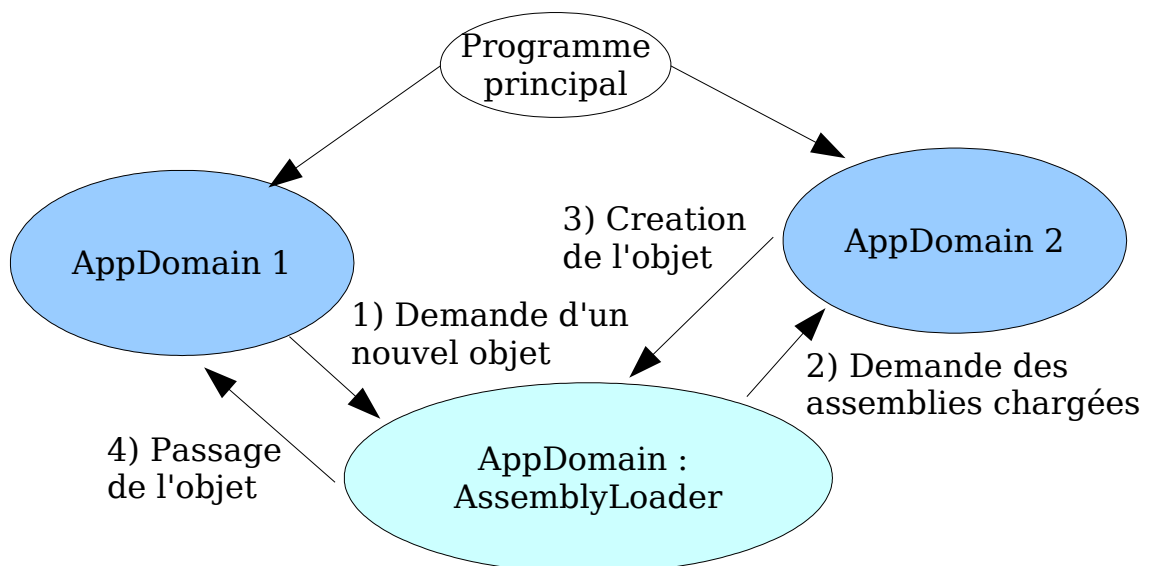


Figure 11 : Principe du chargeur d'assemblies entre plusieurs domaines d'applications

La figure 11 montre l'architecture de cette tentative. Le programme principal crée 3 domaines d'applications : AppDomain 1, AppDomain 2 et l'AssemblyLoader. AppDomain 1 et Appdomain 2 représentent 2 bundles, alors que AssemblyLoader représente le gestionnaire de services de la plate-forme. Lorsque l'Appdomain 1 a besoin d'un objet fourni par l'AppDomain 2, il le demande à l'AssemblyLoader. Celui-ci demande si la classe de l'objet requit est disponible dans Appdomain 2. Si oui, alors ce domaine crée l'objet et le passe à l'AssemblyLoader qui ensuite va le transmettre à l'Appdomain 1.

Ce système fonctionne, il est applicable à OSGi, au prix d'une forte baisse de performances. En effet, comme vu auparavant, la communication entre domaines d'applications ne peut se faire qu'en utilisant .Net Remoting. Donc les performances de l'application s'en trouvent énormément dégradées. D'autre part, il n'est pas possible de partager des objets entre plusieurs domaines sans utiliser cette technologie.

5.3.2) Utilisation d'un seul domaine d'application

Une autre possibilité envisagée, consistait à ne considérer qu'un seul domaine d'application pour toute la plate-forme de services.

Les bundles sont alors représentés comme des assemblies. Un chargeur global permet le passage d'objets de type Type (équivalent aux objet de type class en Java) entre les assemblies. La figure 12 montre le fonctionnement de ce système.

Cette solution est plus éloigné du Module Loader. En effet, la communication entre le chargeur global et les différents modules ne se fait pas par l'intermédiaire d'un chargeur de classe mais par un passage d'un pointeur sur le Type demandé. De plus il n'est pas possible de charger plusieurs versions d'une même assembly. En effet, .Net ne permet pas de charger 2 version d'une même assembly au sein d'un même domaine d'application.

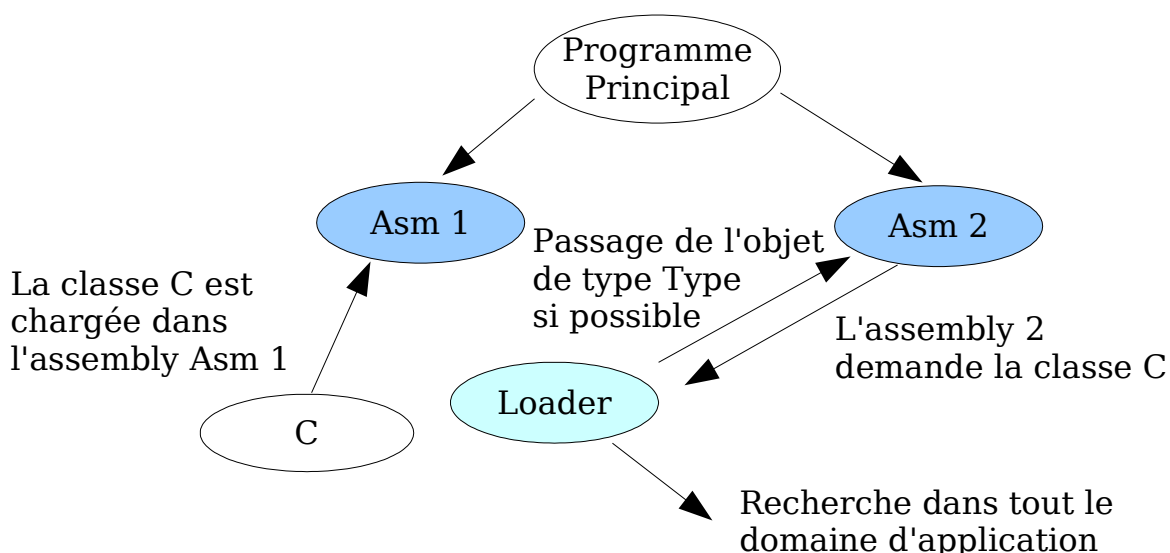


Figure 12 : Principe du chargeur de classes entre plusieurs assemblies

Le problème de cette solution est que, lorsque C est chargée dans l' assembly Asm 1, l' assembly Asm 2 y a accès car ils appartiennent au même domaine d'application. Le chargeur ne fait que passer l'objet Type permettant d'instancier l'objet. Or OSGi rajoute le fait que certaines classes ne doivent pas être exportées. Cette possibilité est exclue en utilisant cette architecture.

5.4) Tentative de court-circuitage du chargeur de .Net

Un autre axe de mes recherches a été de trouver un moyen de court-circuiter le système de chargement de classe de .Net, ce qui permettrait l'utilisation automatique du chargeur personnalisé de façon aveugle pour le développeur.

Le but de ce test était de situer les chargements d'assemblies et de changer le comportement du chargeur de .Net

Pour cela, j'ai essayé la programmation orientée aspects [Weave.Net] afin d'étudier le comportement de .Net lorsqu'il y a chargement d'une assembly et utilisation d'une classe chargée.

La programmation orientée aspect permet d'injecter du code dans les classes de l'application. Ces points d'insertion sont sélectionnés par des propriétés comme les accès à des champs internes d'une classe, les appels de méthodes d'une classe...

Deux logiciels ont été utilisés pour cela : Weave.Net et AspectC#. Mais aucun des 2 outils ne proposent une granularité assez fine pour capturer l'évènement de chargement d'assemblies ou d'utilisation de classe. Il est tout de même possible de capturer les constructeurs des classes. Donc il est possible d'avoir un contrôle sur les classes utilisées et de savoir par qui elles sont utilisées. Mais cela ne suffit pas, car il faudrait aussi contrôler toutes les utilisations de l'objet créé.

5.5) Étude d'une implémentation d'une machine virtuelle .Net

Dans cette partie nous allons étudier en détail une implémentation de la plate forme .Net. Cette étude va permettre d'en comprendre le fonctionnement. Une modification concernant le chargement de classes a été effectuée.

5.5.1) Shared Source CLI

Le Shared Sources CLI (aussi nommé Rotor) est une implémentation portable du CLI faite par Microsoft dont les sources sont disponibles [Stutz2003]. Il ne s'agit pas de sources réelles de la plate forme « officielle » .Net de Microsoft mais d'une version modifiable destinée aux tests et à l'expérimentation. Ces sources sont disponibles depuis décembre 2002 et permettent de comprendre le fonctionnement interne de .Net.

ROTOR est destiné aux chercheurs, aux étudiants ainsi qu'au designers de machine virtuelle. Il se compose de 1,9 millions de lignes de code :

- 1.1 million en C++,
- 600 milles en C#,
- 125 milles en IL (langage intermédiaire)
- le reste en assembleur

Il existe 5.900 fichiers sources, et 9700 fichiers au total.

La compilation de ces sources prend environ 20 minutes sur une machine classique.

Mon étude a surtout concerné le noyau de la machine virtuelle : mscorlib.dll, composé de 867 classes écrites en C++. Pour cela, j'ai utilisé l'environnement de développement Eclipse muni des plugins CDT (gestion des projets C / C++), ainsi que le débogueur Microsoft WinDebug.

De plus, de longues conversations sur les news-groups de Microsoft consacrés au CLR et à ROTOR ont été nécessaires pour comprendre certains points délicats.

5.5.2) Architecture de .Net

L'étude de ROTOR a permis de mettre en évidence la structure réelle des domaines d'applications et des assemblées.

À l'exécution, plusieurs types de domaines coexistent. Tous sont une spécialisation de la classe Base Domain comme le montre la figure 13. Le SystemDomain charge les assemblées dites systèmes c'est à dire nécessaires au fonctionnement de la plate-forme. Le SharedDomain, appelé aussi Neutral Domain, est un domaine dans lequel sont chargées des assemblées que partagent tous les domaines d'application (AppDomain). Les assemblées situées dans ce domaine sont fréquemment appelées Neutral Domain Assemblies.

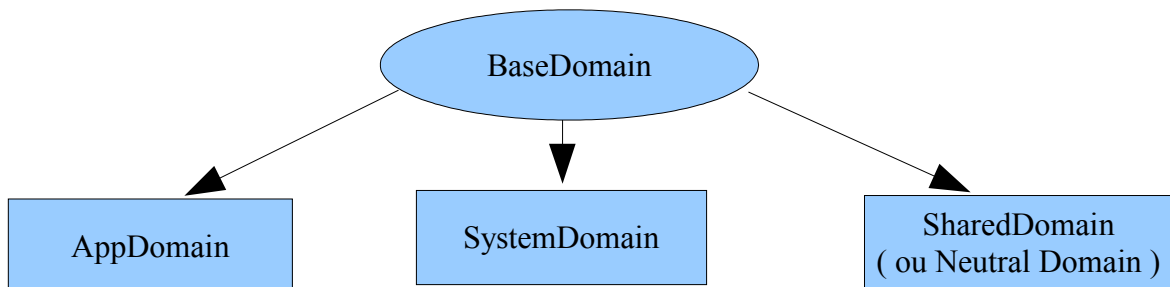


Figure 13 : Hiérarchie des Domaines dans .Net

À l'initialisation de la machine virtuelle, 3 domaines sont créés :

- Un SystemDomain (unique) chargeant le noyau de .Net (sous Windows il s'agit de l'assembly mscorlib)
- Un SharedDomain (unique) partageant cette assembly système
- Un Default Domain dans lequel s'exécute l'application lancée.

D'autres appdomains peuvent bien sûr être créés dynamiquement par l'application s'exécutant dans le Default Domain.

Cette architecture est reprise par l'implémentation « officielle » de la plate-forme .Net. Les actions sur chacun des types de domaines sont très limitées. Il est par exemple impossible de charger une

assembly en temps qu' assembly système sans modifier le code d' initialisation de la machine virtuelle.

5.5.3) Localisation et Chargement des assemblies sur Rotor

Le système de chargement d' assemblies de ROTOR est similaire à celui de la plate forme .Net officielle. Mais l'étude approfondie du code de la machine virtuelle à permis de mieux comprendre les assemblies dite Domain Neutral, c'est à dire chargée dans le Shared Domain.

En effet, les assemblies situées dans le SharedDomain sont aussi disponibles dans le DefaultDomain et ceci sans passer par .Net Remoting. En effet, elles ne sont compilées qu'une seule fois, le code natif ainsi obtenu est partagé par tous les appdomains ayant chargé cette assembly. C'est la seule solution d'avoir du partage de code entre différents domaines d' applications sans passer par les indirections qu'implique une communication inter-processus.

Les assemblies qui ne se trouvent pas dans le Shared Domain sont résolues comme sur la plate forme .Net de Microsoft (figure 14).

ROTOR, tout comme la plate-forme .Net de Microsoft dans sa version 1.1, propose 3 politiques de chargement d' assembly dans le Shared Domain :

- Aucune assembly autre que mscorlib (noyau de la plate-forme), c'est le comportement par défaut
- Seulement les assemblies avec un nom fort
- Toutes les assemblies

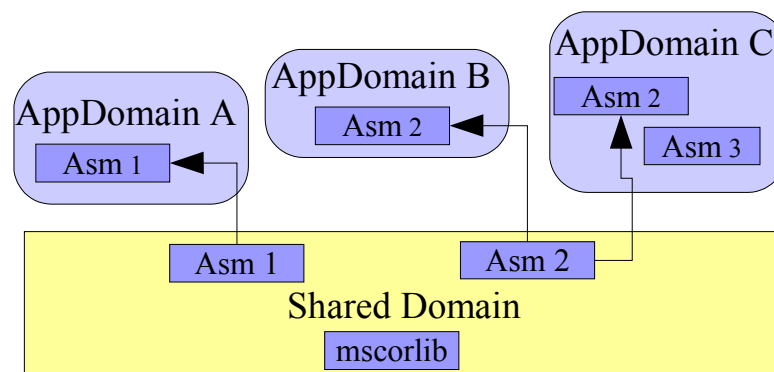


Figure 14 : Shared Domain

La deuxième politique est utilisée pour les applications ASP.Net (pages web dynamiques). Sur un serveur web, il est possible que 2 applications génèrent la même page. Ces 2 applications s'exécutant dans 2 domaines d'applications différents, il n'est pas possible de partager une page générée entre les 2 applications. Avec cette politique, la page est générée une seule fois et les 2 applications partagent cette page. ceci permet de gagner en performance.

Malgré le partage de code, les champs statiques ne sont pas partagés entre tous les domaines d'applications, chaque domaine possède sa propre copie des champs statiques d'une classe. Ceux -ci sont initialisés lors de l'exécution du constructeur de la classe donc dans le domaine d'application.

Il faut considérer le Shared Domain comme une optimisation d'un programme. Microsoft a pour but

de ne plus permettre aux développeurs le choix de chargement en temps qu' assembly partagée. Le CLR choisira lui même si l' assembly doit être partagée ou non. Aujourd'hui cette capacité n'est pas implémentée dans .Net, elle ne devrait pas apparaître avant la version 3 de la plate-forme.

Une autre caractéristique découverte dans Rotor concerne les assemblies multi-fichier. En effet jusqu'à présent, une assembly était stockée dans un seul fichier (.exe ou .dll). Mais il est possible, grâce à la notion de module, de répartir une assembly sur plusieurs fichiers (un fichier par module). Ceci affecte la méthode de chargement de la machine virtuelle. Cette technique est utilisée pour les très grosses assemblies, car l' assembly n'est pas chargée en entier d'un seul coup. Il est possible d'utiliser cette technologie au moment de l'écriture et de la compilation de l' assembly.

Le premier module de l' assembly contient les informations sur le contenu des autres fichiers ainsi que les meta-données des classes. Lorsque l' assembly est nécessaire alors ce fichier est chargé. Ensuite seuls les modules nécessaires sont chargés. Il se peut que certains modules ne soient jamais chargés si ils ne sont jamais utiles. Cette technique permet d'éviter de charger en mémoire du code inutile.

5.5.4) Déchargement de code

Le déchargement de code n'est possible qu'en déchargeant un domaine d'application en entier. Le Default AppDomain n'est pas déchargeable (ceci reviendrait à avoir une machine virtuelle sans programme s'exécutant).

Le déchargement d'un domaine d'application se fait en plusieurs étapes :

1. A l'appel de la méthode AppDomain.Unload, un nouveau thread est créé afin de détruire le domaine
2. Ce thread envoie un signal DomainUnload à tous les Listeners devant prendre en compte l'évènement
3. Le thread gèle la machine virtuelle, il s'agit d'un blocage similaire à celui qui se produit lorsque le garbage collector se lance. Plus aucune application ne s'exécute
4. Le thread renvoie à tous les utilisateurs du domaine en cours de déchargement un DomainUnload Exception, de plus il arrête tous les threads s'exécutant dans le domaine. Plus rien ne peut s'exécuter dans le domaine.
5. La machine virtuelle est réactivée
6. Le Garbage Collector complet (Full Garbage Collector) est appelé, il doit supprimer tous les objets se trouvant encore dans le domaine (ce qui gèle une deuxième fois la plate-forme)
7. Ensuite la mémoire contenant les classes et le code compilé est désallouée
8. Les DLL chargées par le domaine sont déchargées

Le déchargement d'un domaine d'application, prend du temps et bloque la machine virtuelle à 2 reprises; mais il s'agit de la seule manière de décharger du code chargé.

Une solution pour ne décharger qu'une partie d'une application est de charger ces assemblies dans un domaine séparé. Mais cette technique pose 2 problèmes :

- la communication entre le premier domaine et le deuxième doit passer par .Net Remoting
- le déchargement du 2° domaine bloquera l'exécution du premier

5.5.5) Assembly et chargement de classe

L'étude approfondie des assemblies a permis de découvrir que chaque assembly est munie d'un classloader (figure 15), qui a pour but de gérer les classes au sein d'une assembly sous la forme d'objet EEClass (Executive Environment Class).

Chaque EEClass possèdent les méta-données d'une classe (tableau des méthodes ...), elle est aussi liée au code de la classe. Le chargeur de classes de l' assembly charge les classes en provenance des assemblies chargées et place chacune d' elles dans une mémoire nommée Low Frequency Heap appartenant au domaine d'application. Cette mémoire n'est pas explorée par le Garbage Collector de la machine virtuelle. En effet, le Garbage Collector de .Net ne scrute que le High Frequency Heap, situé lui aussi dans le domaine d'application, contenant tous les autres objets. Les classes étant « normalement » des objets à longue durée de vie, elles ne sont pas examinées. Cela implique que dès qu'une classe est chargée, elle n'est supprimée que lorsque le domaine d'application est déchargé.

Le fait que les classes soient stockées dans une mémoire du domaine d'application explique pourquoi toutes les classes chargées sont disponibles à toutes les autres classes du domaine. Lorsqu'un type est demandé, le chargeur de classe de l' assembly examine tout le Low Frequency Heap et renvoie un pointeur sur le type ou la classe demandée.

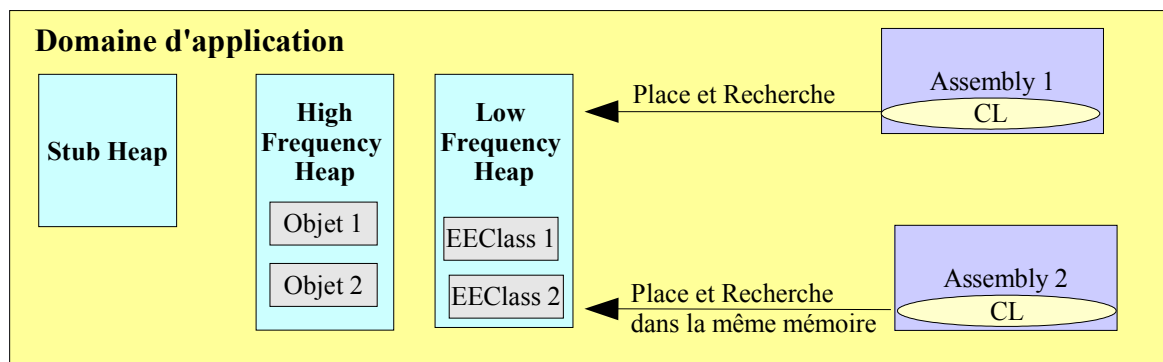


Figure 15 : Système de chargement de classes interne de .Net

5.5.6) Modification du classloader des assemblies

Une modification du classloader interne des assemblies a été réalisée afin d'implémenter une des politiques du Module Loader. L'idée était de permettre uniquement le chargement et l'utilisation des classes contenues dans l' assembly. Il n'était donc pas possible de charger des classes provenant d'autres assemblies (sauf pour les types de base).

Lorsque l' assembly est chargée, elle liste les types qu'elle contient. Suivant le type demandé, le classloader modifié obéit aux règles suivantes :

- Le type provient de l' assembly, alors il est utilisable
- Le type est un type de base (System.*), alors une recherche exhaustive est faite

(elle est réussie forcément étant donné que ces types sont disponibles dans le Shared Domain)

- Le type provient d'une autre assembly, alors une erreur est lancée

Pour cela, une nouvelle classe en C++ a été créée étendant la classe « classloader » (C++). De plus la méthode C# de chargement dynamique (appDomain.load) a été modifiée. Pour finir le constructeur de la classe « assembly » a été modifié pour créer l' assembly avec le classloader désiré.

Il est donc possible d'avoir les 2 classloaders en même temps. C'est au moment du chargement dynamique de l' assembly que le programmeur décide quel classloader l' assembly doit utiliser.

Ce classloader n'est pas encore au point. En effet, il respecte bien la politique énoncée ci-dessus, mais lorsqu'une autre assembly veut charger un type, une erreur est systématiquement levée. D'après Microsoft, cela semble venir d'une corruption du Low Frequency Heap. Je n'ai pour l'instant aucune solution à ce problème, malgré un suivi de l'exécution [SSCLI01].

Chapitre 6 : Perspectives

6.1) Déchargement dynamique d' assemblies

A l'heure actuelle, la machine virtuelle .Net ne propose pas la fonctionnalité de décharger une assembly. Cette capacité n'est pas implémentée mais est en cours d'étude et sera disponible dans la version 4 de la plate-forme .Net.

En effet, plusieurs raisons dans l'architecture du CLR ne permettent pas une implémentation simple d'une telle fonctionnalité :

- Il faudrait vérifier tous les pointeurs ce qui serait trop coûteux. Il faudrait traquer tous les objets pouvant utiliser le code déchargé y compris les objets partagés entre plusieurs domaines d'application (via .Net Remoting)
- La gestion des Domain Neutral Assembly pose problème (Faut il décharger l' assembly dans tous les appDomain, ou seulement dans un seul ?)

Microsoft a commencé à travailler sur cet aspect, mais ce travail n'est toujours pas incorporé aux versions de développement de la plate-forme.

Il est recommandé de bâtir les applications autour des domaines d'applications et d'utiliser .Net Remoting pour la communication entre eux.

6.2) Construction d'application dynamique sur .Net

Le fait qu'il n'est pas possible de décharger du code sans décharger un domaine d'application en entier est un problème important lorsqu'on veut construire une application dynamique basé sur une architecture orientée services. En effet, comme nous l'avons vu au auparavant, il faut pouvoir gérer dynamiquement le cycle de vie des services, or leur retrait est problématique.

Il est tout de même possible de construire des applications orientées services réparties, c'est à dire que chaque service communique avec les autres à travers le réseau. En effet, chaque service s'exécute dans un domaine d'application. Ils ne sont pas forcément sur la même machine. La communication entre eux se fait via .Net Remoting. Lorsqu'un service est retiré, le domaine est simplement déchargé, les services s'étant liés à lui reçoivent un évènement leur signalant que le service n'est plus disponible.

6.3) Implémentation d'un plate-forme OSGi sur .Net

Le fait de pouvoir décharger les assemblies individuellement ne résoudra pas tous les problèmes pour implémenter une plate-forme OSGi sur .Net. En effet, il faudrait aussi pouvoir modifier la portée des classes contenues dans les assemblies chargées, pour pouvoir mettre en place la politique d' import / export utilisée dans OSGi. Comme on l'a vu, cette capacité concerne le classloader interne des assemblies. Or à l'heure actuelle, Microsoft n'a pas en projet de modifier la politique de chargement de classes de .Net.

Chapitre 7 : Conclusion

Nous avons vu tout au long de ce rapport les problèmes rencontrés lors de la création d'une plate-forme de services sur .Net. En effet, le mécanisme de chargement dynamique de code (et de déchargement) de la plate-forme .Net ne permet pas, pour l'instant, d'implémenter une plate-forme conforme aux spécifications OSGi.

Les tests effectués de chargeurs, ne répondent pas aux spécifications de la plate-forme de services. L'étude approfondie d'une implémentation de .Net, nous a montré les mécanismes internes de chargement et de déchargement de code, et les parties à adapter pour pouvoir implémenter un plate-forme OSGi.

Les perspectives de Microsoft, concernant le déchargement de code, ne permettront pas, non plus, la création d'une plate-forme de services OSGi sur .Net, car il n'est pas envisagé de modifier le système de chargement de classes, empêchant ainsi toute mise en place de la politique d'import / export.

Chapitre 8 : Autres documents de travail

Cette partie présente les documents élaborés au cours des premiers mois de ce stage de magistère.

Le chargement de classe en java :

Ce document est une synthèse rapide sur les possibilités de Java concernant le chargement de classes.

<http://www.plop-plop.net/documents/classLoader%20en%20java.pdf>

How does locate the .Net framework assemblies? :

Cet article regroupe les résultats de mes recherches sur le fonctionnement du chargement dynamique de classes sur la plate-forme .Net. Il aborde les différences avec Java et les possibilités offertes par J# et C#.

http://www.plop-plop.net/documents/How_the_.net_framework_locates_assemblies.pdf

Présentation : Classloading on .Net :

Mon travail sur le chargement dynamique de la plate-forme .Net a été présenté à l'équipe Adèle ainsi qu'au Dr Richard HALL, développeur de OSCAR.

http://www.plop-plop.net/documents/presentation_classloading_on_net.pdf

Chapitre 9 : Bibliographie

- [Do2004]** Do, Si-Hoàng - 2004 - *Informatique Répartie et Applications à la Domotique* UNIVERSITE PARIS 8
- [Drayton2002]** Drayton, Peter and Albahari, Ben and Neward, Ted - 2003 - *C# in an nutshell* , O'reilly , ISBN 0-596-00181-9
- [Hall2004]** Richard, Hall and Humberto, Cervantes - 2004 - *An OSGi Implementation and Experience Report*, *IEEE CONSUMER COMMUNICATIONS AND NETWORKING CONFERENCE*
- [Hall2004a]** Hall, Richard S - 2004 - *A policy-driven Class Loader to Support Deployment in Extensible Frameworks* *Component Deployment: Second International Working Conference* , 3083 / 2004
- [Halloway2001]** Halloway, Stuart Dabbs - 2001 - *Component Development for the Java Platform*, Addison-Wesley Pub Co , ISBN 0201753065
- [Cervantes2004]** Humberto, Cervantes - 2004 - *Vers un modèle à composants orienté services pour supporter la disponibilité dynamique*, *Université Joseph Fourier - Grenoble 1*
- [OSGi03]** Initiative, The Open Services Gateway - 2003 - *OSGi Service-Platform Release 3*, IOS Press , ISBN 1 58603 311 5
- [Kirk2002]** Kirk Chen, Li Gong - 2002 - *Programming open service gateways with Java embedded server technology*, Addison-Wesley Longman Publishing Co., Inc. , ISBN 0201711028
- [Endrei2004]** Mark Endrei, Jenny Ang, Ali Arsanjani, Sook Chua, Philippe Comte, Pål Krogdahl, Min Luo, Tony Newling - 2004 - *Pattern : Service Oriented Architecture and Web Services*, IBM INTERNATIONAL TECHNICAL SUPPORT ORGANIZATION , ISBN 073845317X
- [MSDN01]** MSDN - Inside the .Net Framework
<http://msdn.microsoft.com/library/default.asp?url=/library/en-us/cpguide/html/cpconInsideNETFramework.asp>
- [MSDN02]** MSDN - How the Runtime Locates Assemblies
<http://msdn.microsoft.com/library/default.asp?url=/library/en-us/cpguide/html/cpconhowruntimelocatesassemblies.asp>
- [MSDN03]** MSDN - Configuration Files
<http://msdn.microsoft.com/library/default.asp?url=/library/en-us/cpguide/html/cpconconfigurationfiles.asp>
- [Neward2001]** Neward, Ted - 2001 - *Using the BootClasspath*
<http://www.javageeks.com/Papers/BootClasspath/index.html>
- [Neward2001a]** Neward, Ted - 2001 - *Understanding Class.forName()*
<http://www.javageeks.com/Papers/ClassForName/index.html>
- [Ntuba2004]** Ntuba, Jemea - 2004 - *Desing and Implementation of an OSGi Service Architecture for the .Net Platform*, *Free University of Berlin Faculty of Mathematics and Computer Science*
- [OpenWings]** OpenWings - OpenWings Web Site
<http://www.openwings.org/index.html>
- [Oscar]** Oscar - OSCAR : An OSGi framework implementation
<http://oscar.objectweb.org/>
- [SSCLI01]** ROTOR - Logging in the Shared Source CLI

[Stutz2003] Stutz, David and Neward, Ted and Shilling, Geoff - 2003 - Shared Source CLI Essentials, *O'Reilly*, ISBN 0-596-00351-X

[Jini] Sun - Jini Network Technology Web Site
<http://java.sun.com/developer/products/jini/index.jsp>

[Thai2003] Thai, Thuan L. and Lam, Hoang - 2003 - .NET Framework Essentials, *O'Reilly*, ISBN 0-596-00505-9

[Venners1997] Venners, Bill - 1997 - Inside the Java Virtual Machine, *McGraw-Hill Companies*, ISBN 0-07-135093-4

[Weave.Net] Weave.Net - Weave.NET: Language-Independent Aspect-Oriented Programming
http://www.dsg.cs.tcd.ie/?category_id=-26http://www.dsg.cs.tcd.ie/?category_id=-26