Intergiciels systèmes pour passerelles de services ambiants

Stephane Frenot

▶ To cite this version:

Stephane Frenot. Intergiciels systèmes pour passerelles de services ambiants. Génie logiciel [cs.SE]. Université Claude Bernard - Lyon I, 2008. tel-00395181

HAL Id: tel-00395181

https://theses.hal.science/tel-00395181

Submitted on 15 Jun 2009

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

N° d'ordre : 2008-ISAL-0104 Année 2008

Habilitation à Diriger des Recherches

Intergiciels systèmes pour passerelles de services ambiants

À présenter devant

L'UNIVERSITÉ LYON I
Informatique

pour l'obtention de

L'HABILITATION À DIRIGER DES RECHERCHES
spécialité informatique

par

Stéphane FRENOT

À soutenir le 02 Décembre 2008

Devant le jury composé de

Directeurs :	Pr. Stéphane Ubéda,	INSA de Lyon
Rapporteurs :	Pr. Serge Chaumette,	Labri Bordeaux
	Dr. Olivier Festor,	INRIA Loraine Nancy
	Pr. Michel Riveill,	Université Nice Sophia-Antipolis
Examinateurs :	Pr. Didier Donsez,	Université Joseph Fourrier Grenoble
	Pr. Yves Robert,	ENS Lyon
	Pr. Lionel Seinturier,	Université Lille

Travaux effectués au sein du Centre d'Innovation en Télécommunications et Intégration de Services (CITI) de l'INSA
de Lyon et de l'équipe Architecture Réseaux et Systèmes (Ares) de l'INRIA Rhône-Alpes

Table des matières

Glossaire **5**

1 Le contexte de cette habilitation **10**
1.1 La création d'une équipe de recherche 10
1.2 Ce que contient ce mémoire 11

2 État de l'art des intergiciels **13**
2.1 Les intergiciels distribués 14
2.1.1 Intergiciels distribués synchrones 15
2.1.2 Intergiciels distribués asynchrones 17
2.1.3 Vers une approche verticale du middleware 18
2.2 Les intergiciels applicatifs 18
2.2.1 Emergeance des nouveaux langage de programmation . . 18
2.2.2 J2EE : Intergiciel Applicatif 22
2.2.3 Les Intergiciels Système 26
2.3 Une pile d'exécution standard pour environnements contraints et pervasifs 40

3 Extensions aux intergiciels systèmes **43**
3.1 Organisation du chapitre 44
3.2 Distribution pour OSGi 45
3.2.1 Pourquoi distribuer ? 45
3.2.2 Bibliographie complémentaire 47
3.2.3 Résultats et Commentaires 47
3.3 Supervision pour OSGi 48
3.3.1 Pourquoi superviser ? 48
3.3.2 Bibliographie complémentaire 51
3.3.3 Projets et Résultats associés 51
3.3.4 Résultats et Commentaires 53
3.4 Virtualisation pour OSGi 53
3.4.1 Pourquoi virtualiser ? 54
3.4.2 Bibliographie complémentaire 55
3.4.3 Résultats et commentaires 55
3.5 OSGi enfoui 57

3.5.1 Pourquoi embarquer OSGi ? 57
3.5.2 Bibliographie complémentaire 59
3.5.3 Projets et Résultats associés 59
3.6 Sécurisation d'OSGi 60
3.6.1 Pourquoi sécuriser ? 60
3.6.2 Projets et Résultats associés 62
3.7 Conclusion 62

4 Conclusion **65**

A Webographie **68**

B Bibliographie **71**

Glossaire

.NET Environnement d'exploitation d'applications distribuées de Microsoft, page 44

ADSL Asynchronous Digital Subscriber Line, page 14

API Application Programming Interface, page 17

BIOS Basic Input Output System, page 13

CBAC Component Based Access Control, page 61

CORBA Common Object Request Broker Architecture (Architecture commune de négociateur de requêtes objet), page 15

DCE Distributed Computing Environnement, page 15

DCOM Distributed Common Object Model (Modèle d'Objets Communs de Microsoft), page 15

GNU Gnu is Not Unix, page 59

HGL Home Gateway Linux : Projet de spécification d'une passerelle de services en C sous linux, page 56

HTML HyperText Markup Language, page 31

HTTP HyperText Transfert Protocol, page 23

IDL Interface Definition Language (Langage de description d'interface de services), page 16

IETF Internet Engineering Task Force, page 15

IoC Inversion Of Control (Inversion de contrôle entre un conteneur et un contenu), page 21

IP Internet Protocol, page 14

JDBC Java DataBase Connectivity (Protocole d'accès générique java aux bases de données), page 25

JMS Java Messaging System, page 25

JMX Java Management eXtensions, page 27

JNDI Java Naming and Directory Interface, page 25

JSP Java Server Pages, page 20

JSR Java Specification Request, page 31

JTA Java Transaction API, page 25

JTS Java Transaction Service, page 25

MOSGi Managed OSGi : support de gestion pour OSGi, page 49

MUSE MUltiService Everywhere : Projet Européen du 6ème programme IST-6thFP-507295, page 52

MVC Modèle Vue Contrôleur : modèle de programmation séparant la vue et le modèle de données associé, page 47

NFS Network File System (Système de fichiers réseau), page 15

OMG Object Management Group, page 15

OSGi Open Service Gateway initiative (Initiative pour une passerelle de services ouverts), page 33

RFC Request For Comments (Processus de normalisation de l'IETF), page 15

RMI Remote Method Invocation (Invocation de méthode distantes de SUN), page 16

ROCS Remote OSGi Cache Server, page 57

RPC Remote Procedures Call (Appel de procédures distantes), page 15

SGBD Système de Gestion de Bases de données, page 26

SOA Service Oriented Architecture, page 27

SOAP Simple Object Access Protocol (Protocole simple d'accès aux objets), page 15

SOP Service Oriented Programming, page 27

TCP Transport Control Protocol, page 15

URL Uniform Resource Locator, page 30

VOSGi Virtual OSGi : virtualisation de passerelles OSGi, page 56

W3C World Wide Web Consortium (Organisation de normalisation du Web), page 15

XDR eXternal Data Representation, page 15

XML eXtension Markup Langage, page 33

OEM Original Equipment Manufacturer (Equipementier), page 13

PME Petite et Moyenne Entreprise, page 62

URL Uniform Resource Locator, page 52

Liste des algorithmes

2.1 Utilisation d'interface java 21
2.2 Le passage de message dans un langage objet 21
2.3 Inversion de contrôle (IoC) 24
2.4 Exemple : la classe Point 28
2.5 Exemple : une classe Cliente utilisant la class Point 28
2.6 Séparation entre interface et implantation 29
2.7 Chargement explicite de classe 30
2.8 Interface `org.osgi.framework.BundleActivator` 34
2.9 Enregistrement d'un service OSGi 34
2.10 Manipulation d'un service OSGi 35
2.11 Approche événementielle sur les services OSGi 36

Table des figures

2.1 Organisation des langages de programmation 20
2.2 Inversion de contrôle dans un conteneur 24
2.3 Organisation hiérarchique des chargeur de classes Java 37
2.4 OSGi d'un point de vu applications système et applications utilisateur 39
2.5 Pile d'abstraction des programmes 41
2.6 Pile d'exécution embarquée pour OSGi 41
2.7 Services non-fonctionnels de la pile d'exécution utilisée 42

3.1 Chaîne de gestion d'EJB sur une fédération de serveurs 45
3.2 Architecture générale de MOSGi 50
3.3 Organisation en passerelles virtuelles pour OSGi 55
3.4 Architecture fonctionnelle de passerelles de services 56

Liste des tableaux

2.1 Architectures logicielles d'invocation de procédures distantes . . 17

Chapitre 1

Le contexte de cette habilitation

1.1 La création d'une équipe de recherche

Mon recrutement en 1999 au département télécommunications services et usages de l'INSA de Lyon s'est fait pour l'anniversaire de la seconde promotion d'étudiants entrant. Du côté enseignement, les étudiants étaient répartis sur les 2 premières années du cycle ingénieur. Le département était tout neuf et il était nécessaire de monter les cours relatifs à Java pour les 3ème année et les cours relatif aux systèmes distribués pour les 4ème année (l'INSA comporte 5 années). Du côté recherche, au moment du recrutement, aucun laboratoire n'existait, mais l'intention de la direction de l'INSA était de monter une équipe de recherche en Télécommunications associée au département. J'ai été recruté en même temps qu'un collègue spécialiste du traitement du signal. Six mois plus tard, en février 2000, le professeur Stéphane Ubéda a intégré le département dans l'objectif de monter le laboratoire CITI [Citi 2000][1]. Venant de trois horizons orthogonaux, à savoir la compilation et les environnements parallèles, le traitement du signal et le génie logiciel et les systèmes à composants, nous avons décidé de monter une activité de recherche transversale autour de la fourniture de services en réseaux ad-hoc. L'objectif du laboratoire est alors d'étudier les problématiques de livraison de service de bout-en-bout, c'est-à-dire du traitement du signal aux applications utilisateur associées. Le laboratoire CITI s'est étoffé de personnes spécialistes du domaine central, à savoir le réseau, sachant que les deux extrémités de la pile protocolaire étaient étudiées. Après 2 ans d'activité nous avons proposé une équipe de recherche INRIA, ARES [Ares 2002][2]. C'est dans le cadre de cette équipe et du laboratoire CITI que j'ai réalisé la totalité de mon activité de recherche autour de la passerelle de services OSGi. La taille de l'équipe ARES devenant suffisante, Stéphane Ubéda a proposé de

[1] Centre d'Innovations et d'Intégration de Services `http://citi.insa-lyon.fr`
[2] Architecture de Réseaux et de Systèmes `http://www.citi.insa-lyon.fr/team/ares/`

scinder le projet ARES en deux projets indépendants. L'équipe A4RES (Ares 2), reste orientée sur les réseau sans fils avec une forte orientation sur les réseaux de capteurs et le matériel associé, et l'équipe Amazones est plus orientée sur les réseaux ambiants et les logiciels associés.

Je suis un des membres à l'initiative du laboratoire CITI, de l'équipe ARES, de l'équipe AMAZONES et du département Télécommunications Services et Usages. Dans ce contexte, j'ai choisi de ne pas poursuivre mes travaux de recherche antérieurs autour de l'informatisation du dossier médical et de sa représentation textuelle, mais de m'orienter vers un domaine à la jonction entre le système d'exploitation, la supervision de système et le génie logiciel. Je me suis focalisé sur un objet spécifique d'étude à savoir la pile Java/OSGi afin d'en proposer des extensions dans le cadre de passerelles matérielles de services.

1.2 Ce que contient ce mémoire

Ce mémoire résume mon activité de recherche de 2000 à 2008, qui s'est orientée autour des passerelles de services logicielles de type OSGi. L'activité a été financée par 1 projet national de type ACI GRID [Projet DARTS 2002b] sur 2 ans (2000-2002), 1 projet Européen [Projet MUSE 2007] sur 4 ans (2003-2007) et 1 ANR sécurité [Projet LISE 2010] sur 3 ans (2008-2011). Durant ces années j'ai fait soutenir 2 thèses [Masri 2005, Royon 2007] et en suit une autre. J'ai formé et encadré environ 6 stagiaires par an, autour des technologies que nous utilisons, qui se répartissent entre des étudiants de Mastère recherche, de projets de fin d'étude et de stages d'élèves ingénieurs. J'ai également eu le support de 4 ingénieurs de recherche financés par les différents projets. J'emploie encore actuellement 2 ingénieurs.

Mon activité est fortement dirigée par le génie logiciel des intergiciels et se situe à la frontière entre les problématiques issues de la grille de calcul, de l'administration de systèmes, des systèmes d'exploitations et de la sécurité. J'ai pris soin de diffuser et de publier l'ensemble des travaux réalisés dans l'équipe dans des journaux, dans des conférences, dans des workshops mais aussi et surtout dans la rédaction de rapports techniques et par la diffusion de code open-source dans les communautés concernées. Je suis responsable de l'équipe intergiciel du laboratoire CITI qui est structuré en 5 équipes autour des thèmes suivants : Senseurs, Protocoles, Modèles, Intergiciels, Systèmes embarqués.

Ce mémoire reflète une synthèse de mes activités en présentant les différentes extensions que nous avons conçues et intégrées à un intergiciel système pour applications ambiantes. Il est structuré en deux parties et une conclusion. La première partie présente un état de l'art sur les intergiciels et la seconde partie présente les extensions que nous avons étudiées, implantées et diffusées. Ces extensions sont faites dans les domaine suivants :

- **Distribution de services**, on montre comment on étend notre intergiciel système local pour l'exploiter sur des machines distribuées et collaboratives,

- **administration de services**, on montre comment on peut superviser et monitorer des équipements ambiants,
- **virtualisation de services**, on montre comment exécuter des services dans des espaces presque isolés, c'est à dire offrant le support mais pas le contrôle de l'isolation,
- **réduction de services**, on montre comment économiser de la place disque pour l'exécution sur des environnements ne permettant pas d'héberger localement les piles d'exécution standard,
- **sécurisation**, on montre comment identifier et gérer les problèmes de sécurité dans des environnements extensibles.

Les annexes incluent mon dossier scientifique détaillé, une webographie et une bibliographie complète. Tous les sites webs utilisés sont référencés en note de bas de page, dans la webographie et dans la bibliographie générale.

En vous souhaitant une bonne lecture...

Chapitre 2

État de l'art des intergiciels

La définition d'une passerelle de services est une chose complexe, car elle fait la jonction entre deux communautés ou deux mondes qui n'ont pas de raisons initiales de se côtoyer. Historiquement, et avant que MICROSOFT ne propose des contrats OEM, le matériel et le logiciel étaient indissociables. Toutes les couches, de la carte mère aux applications en passant par le BIOS, le système d'exploitation et la pile de communication étaient définies et implantées par la même entité. Quand Microsoft a proposé un contrat de type équipementier, l'entreprise IBM n'y voyait pas d'inconvénient [Goldman Rohm 1998]. Cette indifférence s'explique par le fait que le cœur d'activité d'IBM à ce moment est dédié aux gros systèmes et non aux systèmes situés à la périphérie comme l'ordinateur personnel. Il considère également que la commercialisation d'un système matériel est largement plus rentable que les systèmes logiciels exécutés dessus. La séparation et l'indépendance de commercialisation du matériel et du système d'exploitation a ouvert le marché et la potentialité de l'informatique moderne. Du moment où l'intégralité des éléments d'un système pouvaient être imaginés comme autant de briques élémentaires coordonnées, l'informatique n'était plus contrainte aux systèmes monolithiques, centralisés et mono-fournisseur. Néanmoins l'apparition d'une approche multi-fournisseurs nécessite de définir toutes les interfaces de fonctionnement entre les composants. Le nombre de ces interfaces dans le domaine du système d'exploitation est telle que cette révolution a eu un coût non négligeable sur la qualité et l'interopérabilité des éléments logiciels fournis. En effet, s'il est difficile d'imaginer une voiture équipée de ceintures de sécurité défectueuses, installer un système d'exploitation ne fonctionnant pas parfaitement est d'une part une réalité et d'autre part un fait accepté par la majorité des utilisateurs. De plus, si dans le domaine de l'automobile il existe des organismes indépendants, comme les mines ou les organismes de crash-test, qui garantissent la cohérence du véhicule, il n'existe pas d'approche équivalente dans le domaine du logiciel informatique.

Le monde de l'intergiciel[1] possède une nature intrinsèque complexe du fait

[1] Middleware en anglais, nous utiliserons indifféremment les deux termes.

de la multitude des technologies et des concepts qui y sont associés. Ce chapitre a pour objectif de présenter les différentes évolutions que nous avons observées et nous permettre de placer certains éléments clés des intergiciels. Notre décomposition scinde la progression du middleware en trois phases. La phase des intergiciels distribués, celle des intergiciels applicatifs et celle des intergiciels systèmes.

2.1 Les intergiciels distribués

L'indépendance entre les systèmes terminaux (postes de travail) et les systèmes centraux (serveurs) a nécessité la mise en place du réseau et des couches de communication. Très vite la diversité et l'hétérogénéité des réseaux a poussé la définition d'équipements permettant d'interconnecter deux réseaux "incompatibles", c'est à dire ne parlant pas la même langue. Cet équipement de traduction est une passerelle (ou gateway en anglais), permettant de faire communiquer 2 mondes, comme par exemple les réseaux Ethernet et les réseaux Token-Ring. La notion de passerelle a été intégrée dans la spécification du protocole IP comme étant l'équipement capable de faire "sauter" les paquets de réseau en réseau. D'une manière générale, elle permet de faire transiter l'information aux frontières d'un monde vers un autre monde (passerelle IP, passerelle ADSL...).

Le fait que la passerelle d'interconnexion soit d'une part un équipement incontournable et d'autre part un point de convergence de l'information a rapidement attiré l'attention afin de pouvoir lui faire exécuter d'autres activités. Son rôle de convergence de l'information l'identifie comme le meilleur candidat pour toutes les fonctions de sécurité et de configuration (filtrage de paquets, distributions d'adresses IP). En seconde approche, avec l'augmentation de puissance de ces équipements, la possibilité d'héberger des services plus "applicatifs" est apparue. Les passerelles de services apparaissent avec la livraison de services spécifiques pour l'utilisateur final. Ainsi dans les versions initiales, les fonctions de la passerelle sont limitées à la traduction et au contrôle des communications entre un monde A et un monde B de même nature (réseau). L'hébergement de services permet par généralisation la traduction d'un monde A (réseau) vers un autre monde comme la télévision ou la supervision d'équipements domestiques. Si la traduction de protocoles réseau ne concerne que des spécialistes du réseau, le passage de flux Multicast vidéo vers une chaîne de télévision concerne des spécialistes du réseau, du système d'exploitation et du traitement de l'image.

La mise en œuvre d'un système aussi complexe pour l'hébergement de services à domicile nécessite de repenser en grande partie des couches logicielles (OS, Réseaux, Application) permettant de faire communiquer différentes équipements. Les évolutions constatées sur les systèmes d'exploitations sont focalisés sur l'efficacité du système local et non pas sur son interaction avec le monde extérieur, domaine reservé à la pile réseau. Mais la pile réseau quant à elle, se focalise sur le transport des paquets et non pas sur une vision d'exécution d'applications, domaine reservé aux systèmes d'exploitation. Il faudra attendre la notion d'intergiciels pour avoir des améliorations globales liant les systèmes

d'exploitation, les couches de transport, les équipements, les applications et enfin les utilisateurs.

Dès l'instant où l'information a été échangée entre deux machines hétérogènes, une fonction de traduction est mise en œuvre. L'intergiciel se place entre une application et un système d'exploitation pour permettre la traduction d'une information d'un monde A vers un monde B. Le premier intergiciel est donc la couche logicielle permettant de garantir qu'une information générée sur un micro-processeur big-endian peut être exploitée sur un environnement little-endian ou pdp-endian. Cette première approche a amené l'ajout de fonctions de conversion hton, ntoh[2] dans la pile de transport TCP/IP [Stevens et Wright 1995]. L'intégration de ces fonctions a été directement faite dans la pile de transport et leur exploitation est à la charge du "bon" programmeur. Le problème est que très rapidement il est devenu nécessaire de standardiser les échanges pour les éléments de plus haut niveau comme les tableaux, et les structures de données complexes. L'hétérogénéité des architectures et des systèmes d'exploitation rend l'intégration de fonction de conversion de haut niveau non intégrables au cœur des systèmes. Les librairies de standardisation des types comme XDR [IETF-1832 1995] sont définies indépendamment des systèmes d'exploitation et forment ainsi une première approche des intergiciels, c'est à dire une couche intermédiaire entre le système d'exploitation et les applications afin de fournir des couches standard applicatives. L'interposition de code entre une application et le système d'exploitation est devenu évident et des systèmes comme Corba ou les RPC, ont permis de démocratiser le développement d'applications distribuées. Les intergiciels permettant d'abstraire la couche réseau sont de deux familles. Les intergiciels synchrones, très proches des langages de programmation et les intergiciels asynchrones, plus puissants mais nécessitant de modifier les approches de codage client/serveur classiques.

2.1.1 Intergiciels distribués synchrones

L'évolution des middlewares orientés réseau prend son origine dans les appels de procédures à distances définies par SUN et standardisés par le RFC 1050 [IETF-1050 1988] de l'IETF. Les RPC permettent par compilation d'un descripteur de prototype de fonctions C d'obtenir un code client que le programmeur peut exploiter. Ce code se charge de propager une requête lancée sur une machine cliente locale vers une machine serveur distante. Les RPC ont permi la mise en place d'applications complexes distribués sous Unix comme NFS [IETF-1094 1989]. Les mécanismes des RPC sont le fondement d'un ensemble d'architectures d'invocation de services à distance parmis lesquelles on peut citer DCOM [Eddon et Eddon 1997] de Microsoft, RPC du DCE [3], CORBA de l'OMG [4] et assez récemment SOAP du W3C [5]. Le tableau 2.1

[2] man byteorder
[3] `http://www.opengroup.org`
[4] `http://www.corba.org`
[5] `http://www.w3.org/TR/soap/`

donne un comparatif de ces architectures. Ces mécanismes partagent des caractéristiques communes : une interaction orientée services ; des annuaires ; des outils de génération de code interposé.

Interactions orientées services Pour un code applicatif, exploiter une fonction locale nécessite soit de définir cette fonction, soit de connaître la librairie dans laquelle elle est définie. Dans le cas d'un middleware réseau la fonction (distante dans ce cas) est définie par son interface, c'est-à-dire, soit une signature de fonction, soit un ensemble de fonctions regroupées dans un descripteur. Les descripteurs "classiques" sont par exemple un IDL pour DCOM [6] et CORBA [7], ou une interface Java pour RMI. La notion de service est un ensemble de fonctions qu'un utilisateur peut exploiter localement mais dont les exécutions effectives se font en passant par les couches de transport du réseau, localement ou à distance.

Annuaires Dans un système à base de services, un client doit pouvoir trouver et interagir avec un service distant. La solution classique pour mettre un client et un service en relation est de passer par un annuaire. L'annuaire assure l'association entre un service et un serveur lors de leur inscription, et garantit de fournir leur référence aux clients qui désirent interagir avec eux. Les annuaires de services associent donc une identification d'interface avec une référence exploitable sur un service. Ils représentent également le seul point d'entrée dans le système. Sans annuaire et sans connaissance initiale des services, il est impossible d'interagir avec le middleware. L'annuaire permet donc à un client d'obtenir une référence initiale sur un objet de service. Une fois cette référence obtenue, il est possible d'interagir avec aux travers des interfaces de service.

Outils de génération du code intergiciel Pour qu'un client puisse dialoguer avec un objet de service, il est nécessaire d'interposer entre les deux éléments un certains nombre de fonctions permettant d'absorber l'hétérogénéité des systèmes communiquants. L'hétérogénéité des protocoles réseaux et des langages de programmation utilisés nécessite la mise en œuvre d'une infrastructure d'interposition entre l'appelant et l'appelé afin de standardiser l'ordre des octets, le sens de lecture des tableaux, et l'organisation des structures de données.

Le tableau 2.1 indique les implantations des concepts présentés pour chaque technologie d'appel distant.

Le fait de pouvoir masquer la complexité du réseau à l'aide d'un middleware d'automatisation des appels de procédures distantes a permis d'imaginer la mise en œuvre simplifiée d'applications distribuées. Cependant le principe des appels de procédures distantes mettant en relation synchronisée un client et un service à travers un réseau se révèle problématique dans certains cas d'utilisation. La section suivante présente les approches d'intergiciels distribués asynchrones qui permettent de rendre indépendantes l'activité du client et du serveur.

[6] `http://tinyurl.com/4yhxmk`
[7] `http://www.omg.org/technology/documents/formal/corba_2.htm`

TAB. 2.1 – Architectures logicielles d'invocation de procédures distantes

	Expression de services	Annuaire	Génération de code
RPC	#define	portmapper	rpcgen / XDR
DCOM	msidl	registry windows	Visual tools de Microsoft
CORBA	idl	CosNaming Service	Outils de génération de la plate-forme choisie
RMI	interface Java	RmiRegistry	rmic
SOAP	Webservice	Registry	N'existe pas

2.1.2 Intergiciels distribués asynchrones

Nous avons vu que les intergiciels distribués synchrones présentent un avantage de poids qui est celui de coller avec un modèle de programmation client serveur. Ce modèle de programmation correspond au modèle procédural d'exécution des langages de programmation comme le C ou Java. L'invocation d'un service distribué correspond exactement à l'invocation d'une méthode. Cette pseudo équivalence entre langage de programmation et intergiciels distribués a fait croire qu'il était possible de réaliser n'importe quelle application locale comme une application distribuée en interposant un intergiciel distribué. Cette démarche a été critiquée dans [SCK et Wyant 1994]. La présence d'un réseau par essence *best-effort* et ne garantissant pas les délais entre les clients et les serveurs rend difficile la généralisation de tout type d'application à un système distribué synchrone. Une solution envisagée est d'imposer un intergiciel asynchrone pour le développement d'applications distribuées. Un intergiciel distribué asynchrone dont la famille représentative sont les MOM[8]permettent une relation entre client et serveur qui n'est pas liée par les connexions réseau ou le temps. Mais elle nécessite la mise en place d'un tiers qui se charge de transporter les messages entre les deux entités. Ce message transforme fondamentalement le code du côté client, car celui-ci, une fois qu'il a envoyé sa requête, doit se mettre en attente sur la réponse du serveur. La réponse étant asynchrone, le client doit être libéré pour exécuter d'autres tâches. Cette approche efficace et largement adoptée amène cependant une rupture entre le langage de programmation procédural, par essence client-serveur, et l'intergiciel asynchrone. Cette rupture est concrétisée par une API spécifique qui rend nécessaire un apprentissage lui aussi spécifique du système utilisé. L'apparition des intergiciels asynchrones est liée à celle des grandes applications distribuées large échelle, mais l'approche synchrone reste fortement valable pour des systèmes garantissant une bonne disponibilité et à fortiori des systèmes dont l'intégralité de l'exécution est locale à la machine.

[8]Message Oriented Middlewares

2.1.3 Vers une approche verticale du middleware

Les middlewares réseau synchrones ou asynchrones ont donné naissance à la notion d'architecture orientée services. Dans ces architectures les services sont mis à disposition sur des serveurs par des annuaires, et les applications clientes s'exécutent par composition de services élémentaires. Cette approche n'est pas nécessairement distribuée. Elle permet d'isoler le système d'exploitation hôte des applications. Alors que les premiers sont perçus de manière horizontale car on met en relation différents pairs au niveau du réseau, les intergiciels applicatifs que nous présentons sont plus à voir comme des middlewares verticaux en mettant en relation les différentes couches de programmation et en offrant de plus en plus de niveau d'abstraction aux applications.

2.2 Les intergiciels applicatifs

Les middlewares applicatifs ont pour objectif de simplifier la vision que possède une application des API de bas niveau liées au système d'exploitation et aux problématiques de distribution sur le réseau.

Ces intergiciels sont apparus d'une part avec l'emergeance de l'Internet et d'autre part l'apparition de nouveaux langages de programmations plus faciles d'apprentissage que les langages industriels et académiques comme le C ou LISP. L'apparition d'Internet à forcé les entreprises à passer à l'ère de la communication. Et ce passage devait se faire rapidement. Il était donc nécessaire de définir et de fournir les outils permettant de simplifier ce passage. Cette section présente un historique d'apparition de ces outils. Nous la structurons en 3 phases. La première correspond à l'émergeance de nouveaux langages de programmation facilitant le développement rapide d'applications. La seconde correspond à l'apparition des plates-formes de services comme J2EE ou .Net que nous appelons intergiciels applicatifs. Enfin la troisième phase correspond à l'apparition des intergiciels systèmes, c'est à dire aux plates-formes servant de cœur d'exécution aux intergiciels applicatifs.

2.2.1 Emergeance des nouveaux langage de programmation

Il existe de nombreux langages généralistes de programmation depuis l'apparition du fortran dans les années 50[9] et il est assez complexe de les catégoriser. Nous proposons ici une structuration par rapport à notre expérience passée de programmeur. Les catégories présentées ne sont évidemment pas exhaustives et nous ne cherchons pas à les comparer mais elles nous permettront de justifier l'intérêt que nous portons à Java comme point de convergeance des préoccupations citées.

[9] http://www.oreilly.com/news/graphics/prog_lang_poster.pdf

Les langages industriels Les langages industriels sont ceux qui sont d'une part utilisés dans l'industrie et d'autre part enseignés comme base de l'informatique moderne. Dans cette catégorie, nous pouvons citer le C [Kernighan et Ritchie 2004], le C++ [Stroustrup 2003], Java [Niemeyer et Peck 1998], C# [Hilyard et Teilhet 2008], et si nous remontons dans le temps tous les langages dédiés à des domaines spécifiques comme Fortran pour le calcul ou Cobol pour la gestion. Ces langages "industriels" sont largement utilisés pour tous types de programmes. Ils permettent de comprendre la programmation structurée et offrent une bonne garantie de temps de réponse lors de leur exploitation. La transition entre langage procéduraux comme le C et l'approche objet s'est faite dans les années 95, mais la nécessité d'avoir des langages proches du matériel comme le C font que tous ces langages restent trés utilisés et qu'aucun n'a été délaissé au profit d'un autre.

Les langages interprétés Les langages interprétés ou langages de glu ou langages de script ou langages de prototypage sont les boîtes à outils de tout programmeur. Nous pouvons citer les shells de systèmes d'exploitation, Perl [Schwartz et al. 2006], TCL/tk [Welch 2008], Python [Lutz 2007], Ruby [Thomas et Fowler 2004], ou encore Java et C#. Alors que les langages compilés définissent a priori l'ensemble des fonctions qui seront exécutées au cours de la vie du programme, un langage interprété repose sur un interprète qui "exécute" au fur et à mesure les instructions envoyées. Cette exécution "au fil de l'eau" fait que les types ou structures manipulés peuvent apparaître en cours d'exécution du programme. Cette caractéristique est primordiale pour le cycle de vie d'applications dynamiques, qui s'enrichissent au cours de leur exécution et qui possèdent des durées de vie très longues.

Les langages du Web et des interfaces graphiques L'apparition du Web et d'une manière plus limitée des interfaces graphiques dans les années 2000, ont amené une gamme de langages spécifiques facilitant la réalisation d'applications dédiées aux interactions avec un utilisateur. Parmi ces langages citons Visual Basic [Halvorson 2008], (HyperCard [Goodman 1995] par nostalgie), JavaScript et ActionScript [Moock 2007], PHP [Sklar et Trachtenberg 2006], TCL/tk et Python et enfin Java et C#. Ces langages permettent de fabriquer rapidement une interface graphique interagissant avec un utilisateur pour récupérer ses commandes clavier et lui présenter les résultats. Bien qu'ils reposent tous sur un interpréteur, nous les identifions spécifiquement, car ils illustrent le passage dans le "grand public" des langages de programmation. Grâce à ces langages, pour la majorité très ludiques, de très nombreuses personnes sont capables de monter un site Web ou de faire une petite application de gestion de cave à vins.

A la vue de cette organisation, deux langages apparaissent à la jonction des trois domaines présentés : Java et C#. Nous considérons ces deux langages comme suffisamment proches pour ne pas les distinguer dans une première étape et nous utiliserons le terme de Java pour désigner cette classe de langages.

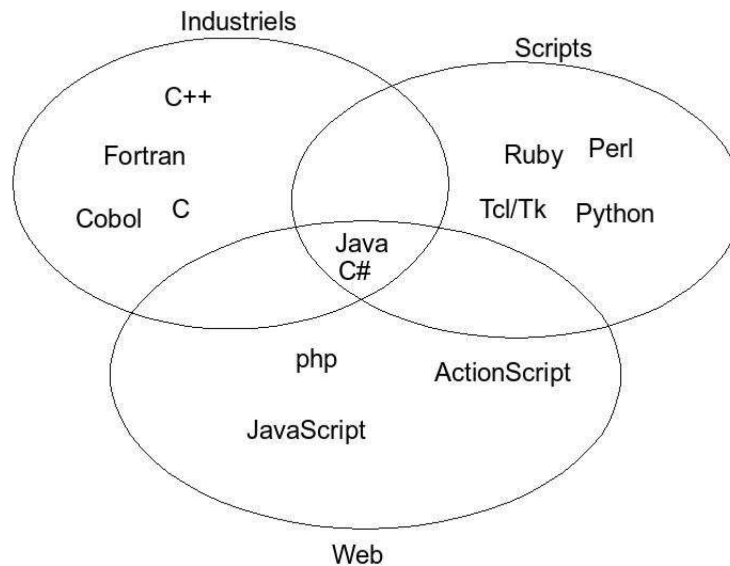

FIG. 2.1 – Organisation des langages de programmation

Java est-il industriel ? Si la bataille du "langage industriel ultime" a fait rage à la fin des années 90 entre Java et le C++, Java reste gagnant sur le nombre de programmeurs, les tendances d'évolution des projets et l'enseignement de base. Pour s'en convaincre il est nécessaire de suivre pendant quelques années des sites d'indices.[10]

Java est-il un langage de script ? Si java n'est pas le meilleur langage pour réaliser un prototype ou programmer rapidement une application, contrairement à Perl, Python ou Ruby, il reste un langage interprété dans la mesure où le byte-code généré est interprété par la machine virtuelle. De plus, la machine virtuelle autorise l'élément fondamental d'extensibilité du code, associé aux langages de scripts, par le mécanisme de chargement dynamique de classes, que nous détaillerons plus tard.

Java est-il un langage du Web ? Java est apparu dans la tourmente du Web lorsque Netscape a intégré une première machine virtuelle Java dans son navigateur. L'objectif est alors d'étendre les fonctionnalités du navigateur par téléchargement dynamique de code appelé Applet. Depuis ces balbutiement Java est certainement le seul langage de convergence et de référence du Web. On le retrouve dans les cœurs d'exécution des grands systèmes d'e-commerce à base de J2EE, dans des langages dédiés de génération de pages Web, par les JSP et les Servlets [Bates et Basham 2008], et enfin dans les navigateurs pour exécuter

[10] `http://www.tiobe.com/index.php/content/paperinfo/tpci/index.html`

du code client comme imaginé initialement.

Nous présentons dans cette section les éléments clés principaux qui font de Java le meilleur candidat pour la suite de notre activité.

Langage Objet Java est avant tout un langage de programmation orienté objet. Il possède donc trois caractéristiques qui le distinguent d'autres langages de programmation : l'abstraction par les classes, l'encapsulation, le passage de messages. L'abstraction par les classes permet de définir le comportement des objets fabriqués dans un programme. L'abstraction définie dans Java est une abstraction simplifiée par rapport aux langages objets précédents. Elle permet de définir une seule classe parente dans une relation d'héritage et ne permet pas la surcharge d'opérateur. Elle offre le principe d'interface, qui permet de regrouper des fonctions dans une entité perçue comme un type. Ainsi le code de l'algorithme 2.1

Algorithm 2.1 Utilisation d'interface java

```
Translatable t=new ObjectGraphique();
```

permet de percevoir l'objet instancié `t` de type `ObjectGraphique` selon une facette spécifique regroupant ses fonctions de *translation* (interface translatable). Cette fonctionnalité est au cœur du polymorphisme des objets Java. La nature de l'objet est donnée par sa relation d'héritage, mais il est toujours possible de lui adjoindre des facettes de comportement par le mécanisme d'interfaces. Ce mécanisme est au centre de l'inversion de contrôle (IoC) présentée plus loin.

L'encapsulation est le second concept véhiculé par un langage de programmation Objet. Elle permet de masquer des comportements internes des objets pour ne voir que leur interface de service externe (méthodes non privées de la classe).

Le passage de messages permet enfin à tous les objets de communiquer entre eux. La syntaxe communément admise du point '.' permet d'imaginer l'envoi d'un message entre un objet source et un objet destinataire.

`Algorithm 2.2 Le passage de message dans un langage objet`

```
Hello h=new HelloImpl();
h.sayHello();
```

Dans l'algorithme 2.2 le message SAYHELLO() est envoyé à l'implantation HELLOIMPL par son interface HELLO. Ces trois éléments sont les fondamentaux de la définition d'un langage objet. L'approche objet permet principalement une

modélisation propre et évolutive des programmes. Le second point essentiel du langage Java est qu'il repose sur une machine virtuelle.

Java : une machine virtuelle logicielle Java repose sur un environnement d'exécution de type machine virtuelle. Elle émule un système d'exploitation générique et standard permettant l'exécution isolée d'applications. Cette exécution se fait par l'interprétation du bytecode des classes compilées à partir du code source initial. La machine virtuelle est également vue comme une bibliothèque standard de fonctionnalités. En effet, les classes standards de Java sont une des forces du langage car elles permettent l'exploitation de classes largement éprouvées à travers leur utilisation dans de nombreux programmes.

L'utilisation d'une machine virtuelle offre les avantages suivants :

- L'isolation des accés mémoire, par un mécanisme de gestion automatique. Elle permet une certaine liberté pour le développement d'applications, en automatisant l'allocation et la des-allocation des zones allouées.
- La convergence des API, regroupe dans une même bibliothèque standard des fonctions de gestion de flux de données, d'internationalisation, de programmation réseau et de programmation multi-threadée. Elle permet d'écrire toutes sortes de programmes, sans avoir à recourir à des bibliothèques spécifiques externes.

La machine virtuelle reste un environnement interprété, ce qui lui permet d'ajouter de nouveaux types au fur et à mesure de son cycle de vie. Les types sont chargés à la demande par des mécanismes de chargeurs de classes dont le rôle est de charger les classes en mémoire au moment de l'instanciation des objets et éventuellement de les supprimer lorsque plus aucune instance n'existe. Ce mécanisme de chargement est entièrement ouvert, c'est-à-dire qu'il est possible de charger des nouveaux types en cours d'exécution. Cette fonctionnalité est l'avantage fondamental de Java sur les autres langages compilés.

Java possède dès sa conception de nombreux avantages par rapport aux autres langages existants. La plupart de ces avantages peuvent être perçus comme des inconvénients par les défenseurs des autres langages. Java est interprété donc il est plus lent qu'un langage compilé et exécuté sur la machine native, l'isolation mémoire est un frein à l'écriture de code bas niveau, l'objet est beaucoup moins expressif qu'un langage fonctionnel, etc. Si l'ensemble de ces affirmations est vrai, il n'existe actuellement pas de langage équivalent aussi polyvalent. Java a été initialement présenté comme le langage du Web permettant l'exécution d'Applets dans les navigateurs. La société Sun propose une exploitation du langage Java dans le cadre de serveur d'applications. C'est à dire de systèmes intégrés permettant le développement, le déploiement et l'exploitation d'applications. Cette approche correspond aux intergiciels applicatifs, car ils permettent la mise en œuvre simplifiée d'applications.

2.2.2 J2EE : Intergiciel Applicatif

La plate-forme Java 2 Enterprise Edition a pour objectif de déployer des applications Web dans un délai et une complexité raisonnables. Elle apparaît

dans les année 2000, où le Web devient une réalité mais où la mise en œuvre d'une application professionnelle Web demande des ressources spécifiques que la majorité des entreprises ne possède pas. En effet pour avoir une application dans un délai raisonnable, il est nécessaire de manipuler un certain nombre de concepts transversaux que les entreprises ne maîtrisent pas encore à ce moment là :

- Interaction Web, comment générer facilement des pages Web dynamiquement, comment analyser les informations saisies dans les formulaires et générer un dialogue utilisateur au dessus du protocole HTTP.
- Interaction sur les bases de données, comment interagir et garantir le stockage des données de la manière la plus simple.
- Gestion de transactions, comment savoir si un serveur est en faute ou s'il met simplement plus de temps que d'habitude pour répondre.

Si les fournisseurs de gros systèmes de l'époque comme Oracle et IBM savent proposer des solutions de bout en bout, des interfaces utilisateurs au stockage de données, ils reposent sur des protocoles et des langages propriétaires spécifiques (SQL*net, SQL*forms, SQL*... pour Oracle) et ne favorisent pas réellement la transition vers les interfaces Web déconnectées. La solution est alors de définir sa propre architecture technique spécifique. La plate-forme J2EE de Sun est alors une proposition de middleware pour l'exécution d'applications orientées Web de bout en bout, permettant d'une part la standardisation des développements et la réduction des délais et des coûts de développement de ce type d'applications.

La plate-forme J2EE repose sur un certain nombre de concepts fondamentaux que nous soulignons ici. Comme nous l'avons précisé pour Java, nous ne détaillerons pas son fonctionnement car celui-ci peut être trouvé dans de nombreux ouvrages et sites Web. Nous ne nous attarderons pas non plus sur les détails liés à telle ou telle version de telle ou telle spécification. Nous essayons d'avoir une vision d'ensemble de l'architecture.

En premier lieu, l'intégralité des éléments de la plate-forme s'exécutent sur une machine virtuelle Java et repose sur un ensemble d'API J2EE fournies par Sun. HTTP, RMI et SOAP sont les protocoles de communication retenus pour la communication entre les différents éléments.

L'inversion de contrôle (IoC) [Fowler 2004] [11]

L'inversion de contrôle consiste à ce qu'un objet "s'offre" au pilotage d'un autre. Le principe est que l'objet "piloté" fournit des méthodes dites crochets (*hooks* en anglais) afin que le pilote puisse le contrôler ou le notifier au moment adéquat. Par exemple dans le code du listing 2.3, l'objet de la classe MyBean présente une interface, ou facette, de comportement de passivation. Cette inversion de contrôle peut se faire par différentes techniques.

[11] `http://www.martinfowler.com/articles/injection.html`

Algorithm 2.3 Inversion de contrôle (IoC)

```
public class MyBean implements PassivationBehavior {
  public void passivate(){
    System.out.println("Hello I will be passivated \\
      in a few seconds");
    //Closing sessions
  }
  public void activate(){
    System.out.println("Hello I have just been \\
      woken up, ready for work");
    //Starting connexions
  }
}
```

Une approche simple et directe est d'y voir une relation conteneur/contenu. Le conteneur déclanche les inversions de contrôle et le contenu doit remplir les contrats imposés.

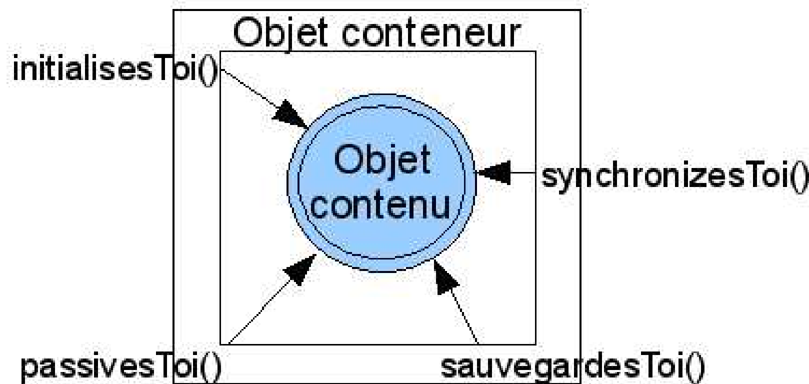

Fig. 2.2 – Inversion de contrôle dans un conteneur

L'inversion de contrôle permet de contraindre les objets sur des comportements standards identifiés par des facettes qu'il doit satisfaire. Dans le langage Java, elles sont imposées par des interfaces.

Un concept de développement associé à l'inversion de contrôle est celui de la séparation des préoccupations (separation of concerns). Il consiste à clairement identifier et capturer chaque activité dans une interface. L'interface reste simple et ne se préoccupe que de la fonctionnalité concernée.

L'inversion de contrôle est donc l'outil fondamental pour contraindre ou garantir le comportement des objets à l'intérieur d'un conteneur. Par exemple, un conteneur peut automatiser la sauvegarde de l'objet dans une base de données, sérialiser l'objet en cas d'inactivité du client, empêcher la fabrication d'une nouvelle instance si le nombre dépasse un seuil dans le conteneur, automatiser la génération de traces d'exécution, etc. Chaque comportement ainsi isolé peut et doit alors être capturé dans une préoccupation et contrainte par une interface Java, ces comportements peuvent être percus comme des facettes suplémentaires aux objets.

La plate-forme J2EE spécifie un certain nombre de facettes qu'un serveur d'applications doit fournir. Parmi les services standards il y a entre autres : un service de nommage conforme à la spécification JNDI [Specification 1998c][12], un service d'interaction avec les bases de données relationnelles JDBC [Specification 1998a][13], un service de gestion de transactions JTA/JTS [Specification 1998d][14], un service standard de messagerie asynchrone JMS [Specification 1998b].[15]

Les archives de déploiement Le second élément d'amélioration apporté par la plate-forme J2EE est le packaging d'applications dans des archives de déploiement. L'intégralité des classes, des ressources, des pages Web statiques et des scripts de génération de pages Web dynamiques sont regroupées dans une archive qui est présentée au moment de l'installation de l'application sur la plate-forme. Ce mode de fonctionnement présente de nombreux avantages :

- reproductibilité des installations : chaque installation repose sur une archive clairement packagée,
- indépendance des fournisseurs plates-formes cibles : le package peut être installé sur n'importe quelle plate-forme compatible J2EE,
- isolation et focalisation sur des services précis : le package peut être simplifié pour contenir une librairie de service spécifique ou ne réaliser qu'une tâche spécifique d'un gros système,
- approche déclarative des interactions avec les services standards externes.

Cette notion d'archive de déploiement est une brique fondamentale des applications à base de conteneurs. Elles sont présentées à l'entrée du conteneur qui en vérifie les caractéristiques et autorise ou non son installation. Un peu à contre-courant de la dénomination standard, nous appelons ces archives des composants, car ce sont des briques unitaires (composants) d'une architecture globale[16].

La plate-forme J2EE est rapidement devenue incontournable pour toute application professionnelle désirant avoir une visibilité sur Internet. De plus, le

[12] `http://java.sun.com/products/jndi/docs.html`
[13] `http://java.sun.com/javase/technologies/database/`
[14] `http://java.sun.com/javaee/technologies/jta/index.jsp`
[15] `http://java.sun.com/products/jms/`
[16] Cette notion de composant est en porte-à-faux avec la notion d'application composée ou à composants dans laquelle les éléments communiquent entre-eux par composition.

caractère interprétatif et dynamique de la machine virtuelle permet aux applications conformes J2EE d'avoir des cycles de vie très longs sans redémarrage des systèmes mais par évolution constante des composants déployés.

L'utilisation des plates-formes de services J2EE a également segmenté le marché du développement d'applications logicielles. Jusque dans les années 2000, les applications étaient soit développées de manière ad hoc, soit développées sur des environnements avancés comme ceux d'*Oracle* cités auparavant. L'apparition de la plate-forme J2EE permet de devenir indépendant des architectures clé en main. Les nouveaux acteurs comme *IBM* ou *BEA* proposent une alternative standardisée d'exploitation d'applications. D'un point de vue économique, il ne s'agit que d'un report des coûts liés à la gestion des données dans les SGBD vers des coûts liés aux traitements applicatifs. Cependant l'apparition rapide de serveurs d'applications open-source et gratuits comme *JBoss*[17] et *Jonas*[18] ont permis une démocratisation de l'architecture J2EE, la rendant ainsi quasiment incontournable dans les architectures de commerce en ligne.

Mais si l'architecture J2EE est incontournable pour le support d'exécution aux applications liées à l'Internet, elle reste figée dans une spécification contraignante et fermée. Elle impose le comportement des conteneurs pour l'IoC, le type de conteneurs offerts (Web et EJB) et les services externes standards fournis. Cette standardisation est un frein à certaines évolutions, et soulève une question de fond : "quelle est la plate-forme d'exécution des serveurs d'applications J2EE ?"[19]. Si les approches par déploiement de composants logiciels et par inversion du contrôle sont parfaites pour le développement d'applications de haut niveau, pourquoi ne conviendraient-elles pas pour développer ces mêmes serveurs, des machines virtuelles ou des systèmes d'exploitation ?

2.2.3 Les Intergiciels Système

Les intergiciels systèmes sont des intergiciels qui se placent au dessus des systèmes d'exploitation en imposant des contraintes minimales pour les applications exploitées au dessus, mais permettant d'améliorer leur processus de développement, de déploiement et d'exécution. Cette section a pour but de présenter certains concepts fondamentaux de ces systèmes.

Il est donc envisageable de concevoir des intergiciels sans service standard imposé mais fournis en fonction du contexte d'exploitation. Si le contexte est un intergiciel applicatif, les services sont par exemple des connecteurs sur les bases de données, si c'est un système d'exploitation ce sera la pile de transport, si c'est une machine virtuelle se sont les classes standards. Ce type d'intergiciel existe

[17] `http://www.jboss.org`

[18] `http://jonas.objectweb.org`

[19] Une question similaire se pose concernant Java. Pourquoi Java n'est-il pas écrit en Java. La réponse vient en partie de nos explications sur le langage. Java est en effet un excellent choix pour ses qualités générales, mais dans le cas de machines virtuelles Java, l'implantation est très spécifique et précise, un langage très ouvert est alors inutile et non performant. On lui préférera certainement le C ou le C++.

depuis quelques années ; il sert de cœur d'exécution à certaines plates-formes J2EE, et nous l'appelons intergiciel Système.

Les intergiciels système permettent la mise en place d'applications complexes à partir de combinaisons de services. Les services sont *packagés* dans des unités d'activation et de déploiement et leur cycle de vie est géré par un contrôleur. L'origine des plates-formes de services pour Java remonte à trois plates-formes de références : JMX [JSR-3 2000], Avalon [Apache Software Foundation 1995] et le framework OSGi [OSGi Alliance 2002]. Elles seront plus tard suivies par de nombreux projets comme Spring [Arthur et Azadegan 2005], Pico [pic 2007] et Nano conteneur [nan 2008] ou encore Guice [gui 2008].

Les principes fondamentaux pour ces passerelles sont liés au génie logiciel et aux propositions d'évolution de la programmation objet vers la programmation orientée services[20]. Nous donnons ici les quelques modèles de conception essentiels[21] des intergiciels système.

L'inversion de contrôle (IoC) et la séparation des préoccupations (SoC) Ce sont des concepts déjà présentés précédemment, et qui se résument selon les deux points suivants :
- Une application est contrôlée du haut vers le bas dans une relation contrôleur/contrôlé.
- Une comportement isolé par une facette ne fait qu'une seule chose mais le fait (très) bien. Si une facette regroupe deux comportements, on extrait un des deux comportements dans une nouvelle facette.

La séparation d'une API (d'une interface) de son implantation Cette séparation permet de dissocier le cycle de vie d'un client du service qu'il exploite. L'objectif de ce *pattern* est de pouvoir remettre en cause une implantation spécifique sans avoir à recompiler un client utilisateur. La suite du paragraphe va expliquer ce principe à partir de quelques exemples de code Java. On suppose qu'on a initialement une implantation permettant de gérer une classe Point, représentant un point dans l'espace décrit dans l'algorithme 2.4.

[20]A clairement différencier des architectures orientées services. SOA !=SOP
[21]`http://www.picocontainer.org`

Algorithm 2.4 Exemple : la classe Point

```
public class Point{
  int x, y;
  public void deplace(int dx, int dy){
    this.x+=dx;
    this.y+=dx;
  }
  public String toString(){
    return "Je suis un Point aux coordonnées : " \\
                  +this.x+","+this.y;
  }
  ...
}
```

L'implantation de la fonction est fausse car elle déplace le point de la valeur `dx` pour la coordonnée `y`.[22]

Algorithm 2.5 Exemple : une classe Cliente utilisant la class Point

```
public class Client{
  public static void main(String [] arg){
    Point p=new Point();
    p.deplace(10,5);
    System.out.println("==>"+p);
  }
}
```

Dans cet exemple si on corrige le serveur, il est nécessaire de recompiler le serveur et d'arrêter et relancer le client associé. Un premier modèle que le programmeur doit suivre est de capturer le service dans une interface. Le code serveur est alors séparé en une interface et une classe d'implantation comme présenté dans l'algorithme 2.6.

[22] Le code client (algorithme 2.5) utilisant cette classe affichera le point aux coordonnées (10, 10) alors qu'il devrait être aux coordonnées (10,5).

Algorithm 2.6 Séparation entre interface et implantation

```
public interface Point{
  public void deplace(int x, int y);
}

public class PointImpl implements Point{
  public void deplace(int x, int y){
    this.x+=dx;
    this.y+=dy;
  }
}

public class Client2 {
  public static void main(String [] arg){
    Point p=new PointImpl();
    p.deplace(10, 5);
    System.out.println("==>"+p);
  }
}
```

Le programmeur de la classe **Client2** peut maintenant avoir le choix entre plusieurs implantations du service Point. Mais si l'implantation est mauvaise il est toujours nécessaire de recompiler l'implantation et le client d'utilisation.

Afin d'avoir une indépendance totale entre le client qui utilise l'instance et sa classe d'implantation, il est nécessaire de passer par un mécanisme d'indirection pour le chargement de classe. Plus particulièrement celui permettant de piloter par programme les chargeurs de classes afin de pouvoir fabriquer une instance par la chaîne de description du nom de la classe. Ainsi le code de l'algorithme 2.7 permet d'obtenir une instance de la classe **test.PointImpl** en indiquant la chaîne de caractères identifiant la classe.

Algorithm 2.7 Chargement explicite de classe

```
public class Usine{
  public static Point getImplementationOfPoint(){
    Class clazz=Class.forName("test.PointImpl");
    Object o=clazz.newInstance();
  }
}

public class Client{
  public static void main(String [] arg){
    Point p=(Point)Usine.getImplementationOfPoint();
    p.deplace(10,5);
    System.out.println("==>"+p);
  }
}
```

Dans cet exemple de code, nous remarquons que le client et l'interface **Point** sont liés par l'usine de fabrication d'instances. Si une implantation est modifiée ni le client ni l'usine n'ont besoin d'être recompilés. L'usine est liée à l'implantation par la chaîne de caractères ``test.PointImpl''. On obtient donc une indépendance, à l'interface de service près, entre client et implantation de service. Il est donc imaginable d'avoir un cycle de vie quasiment indépendant entre un client et un service. Cependant, dans l'exemple précédent, il faut relancer la machine virtuelle cliente pour prendre en compte une nouvelle implantation. Si on veut une indépendance en cours d'exécution de la machine virtuelle, il faut encore apporter les améliorations suivantes :

- L'implantation doit pouvoir être substituée en cours d'exécution. Dans l'algorithme 2.7 la classe est chargée par les chargeurs de classes standards de la machine virtuelle. Ces chargeurs reposent sur une spécification statique[23] des chemins de classes disponibles. Ce chemin est spécifié dans la variable d'environnement CLASSPATH. Java permet d'étendre ces chargeurs de classes par des URL pointant sur des classes chargées à distance. Dans ce cas le système récupère de nouvelles implantations au cours de l'exécution du client. La modification à apporter se situe dans la méthode `getImplementationOfPoint` de l'usine de l'algorithme 2.7.
- Le client doit posséder des points de synchronisation afin de pouvoir redemander de nouvelle implantations. Il existe deux approches possibles : soit le client demande explicitement une re-synchronisation avant un appel de fonction (recherche contextuelle), soit il réside dans un conteneur qui par inversion de contrôle lui injecte une nouvelle dépendance dès qu'elle est disponible (injection de dépendances) [Fowler 2004]. Dans le premier cas le client fait des appels réguliers à la fonction `getImplementationOfPoint`,

[23] avant lancement de la machine virtuelle

dans le second cas le code du client est contraint dans une fonction "crochet" qui est invoquée lors d'une nouvelle injection.

- Lors de la remise en cause d'une implantation, le client doit pouvoir fournir l'ancienne référence et une nouvelle référence doit lui être fournie. Ainsi l'usine doit pouvoir extraire les attributs internes de la première référence pour les réinjecter à l'identique dans la seconde. Comme l'usine, du fait qu'elle est générique, ne peut pas invoquer de constructeur spécifique (la méthode `newInstance` est sans paramètre) une interface spécifique doit permettre de positionner ces paramètres.

En mettant en place ces quelques éléments algorithmiques dans les code d'usines et dans les conteneurs d'exécution des codes clients, il est possible d'obtenir des applications à composants dont les cycles de vie sont d'une part indépendants entre client et services fournis et d'autre part peuvent être maintenues en activité permanente puisqu'il n'est plus nécessaire de les redémarrer ou de les recompiler lorsqu'une nouvelle fonctionnalité est disponible.

Programmation orientée composants La programmation orientée composants reflète deux aspects ; cela correspond d'une part au fait qu'une application soit faite de composants atomiques et d'autre part au fait qu'une application soit réalisée à partir d'une composition de services élémentaires. Nous ne retenons que la première définition et le fait qu'un composant se présente à une plateforme de services introduit une prè-phase d'admission avant l'exécution réelle. Cette phase permet de gérer de nombreuses conditions initiales. Par exemple un composant peut déclarer de futures dépendances envers des services standards, on peut contrôler ses droits d'accès, ou encore mettre en cache les données dont il va avoir besoin. Si certaines de ces préconditions ne sont pas satisfaites, le composant est bloqué par le contrôleur.

L'inversion de contrôle, la séparation interface/implantation et la programmation orientée composants a permis la définition de sur-couches à Java pour l'exécution d'applications complexes. Plusieurs projets sont apparus reprenant ou définissant certains éléments présentés précédemment.

JMX (Sun, 1998) [JSR-3 2000] La spécification proposée par Sun pour la supervision d'applications est issue du 3ème appel à spécification de Sun (JSR) [24] et date de 1998. Elle représente donc une problématique qui a été identifiée dès les débuts de Java. Dans les grandes lignes, elle permet de définir des composants de supervision d'applications java. Un composant est défini par une interface et une implantation associée. Le fait d'utiliser une interface standard Java pour décrire l'objet *"supervisé"* permet de fabriquer automatiquement une console de supervision. En connaissant la liste des méthodes d'un service il est possible de générer aussi bien des pages HTML de présentation du service qu'un client d'interrogation RMI complexe. Les services ainsi décrits s'inscrivent dans un agent de supervision qui maintient dans un annuaire l'association entre une

[24] On est actuellement à la 927ème. `http://jcp.org/en/jsr/all`

interface et son implantation. JMX a été initialement imaginé pour entrer dans le noyau de supervision de routeurs. Mais a également été utilisé comme cœur d'exécution de la plate-forme J2EE *JBoss*. Les différents éléments de la plate-forme (serveur de composants métier, de pages Web, d'annuaire, de connexion aux bases de données) sont implantés par un composant JMX[25]. Cette approche permet une structuration d'une architecture complexe comme J2EE en composants atomiques.

Avalon (Apache, 1998) Avalon est un projet Apache pour la spécification d'applications complexes par composants. Avalon repose fortement sur l'inversion de contrôle des composants et sur un conteneur qui gère l'injection des dépendances entre composants. Les cibles initialement pressenties étaient les applications volumineuses Java. C'est à dire l'ensemble des projets Apaches. Avalon devait servir de cœur d'exécution d'intergiciels applicatifs comme Jserv [Apache JServ 1995][26] (serveur Web), Cocoon [Apache Cocoon 2005][27] (environnement de publication Web). Un composant Avalon suit un cycle de vie précis lors de son intégration dans un conteneur. Le conteneur est l'entité qui va imposer une inversion de contrôle sur le composant et peut être implantée selon différentes approches : classe conteneur, injection de code, programmation par aspects, programmation par annotation... Quelque soit l'implantation choisie, l'inversion de contrôle imposée au composant passe par les phases du cycle de vie suivantes :

1. `LogEnabled` : un système de trace est associé au composant
2. `Contextualizable` : le composant peut récupérer des informations de contexte
3. `Serviceable` (initialement Composable) : les composants de services que le composant veut utiliser sont récupérés
4. `Configurable` : chaque service associé est configuré
5. `Initializable` : chaque service reçoit une notification pour s'initialiser
6. `Startable` : le composant est démarré

Un composant Avalon déclare les interfaces auxquelles il veut répondre. Par exemple si un composant ne dépend d'aucun autre composant, il n'implante pas les interfaces Serviceable et Configurable. Le conteneur n'invoque dans l'ordre du cycle de vie que les interfaces présentes dans le composant. D'autres interfaces sont disponibles une fois que le composant est démarré :

[25] Il faut bien comprendre que JMX a une double vie. C'est d'une part un protocole et un système de supervision d'applications Java intégrés aux routeurs. Mais c'est également un cœur d'exécution au centre d'un intergiciel applicatif comme JBoss. Donc pour JBoss JMX est utilisé dans 2 contextes différents. D'une part dans son cadre d'utilisation pressenti pour remonter des informations de supervision de l'intergiciel applicatif, mais également comme moteur d'exécution. Ce double comportement a quelque peu perturbé le potentiel que JMX possédait.

[26] http ://archive.apache.org/dist/java/jserv/

[27] http ://cocoon.apache.org/

- `Suspendable` : invoquée lorsque le composant doit être temporairement arrêté
- `Recontextualizable, Recomposable, Reconfigurable, Reparameterizable` : invoquées lorsque le composant remet en cause une de ces activités
- `Disposable` : invoquée lorsque le composant est supprimé

Dans l'approche Avalon, tous les composants sont des composants Avalon exécutés dans des conteneurs Avalon. Ils sont difficilement exploitable en dehors de ce contexte. Les applications sont décrites par des descripteurs externes qui indiquent dans des fichiers XML les relations entre composants, afin que le conteneur puisse automatiser le lancement de l'application. Il existe un certain nombre de conteneurs issus des travaux initiaux d'Avalon comme Loom [Loom 2005][28], Excalibur [Excalibur 2005][29] ou Keel [Keel 2005][30]. Du fait de divergences sur la manière d'aborder cet intergiciels le projet a été arrêté, segmenté et relancé dans de nombreux autres projets équivalents.
[31]

OSGi (OSGi Alliance, 1999) [OSGi Alliance 2002] Parallèlement au projet Avalon, Sun a débuté en 1998 un autre projet d'intergiciel système pour passerelles domestiques de services. L'alliance OSGi définit une architecture de livraison de services décentralisée pour l'installation et l'exécution de composants logiciels sur des passerelles de type set-top box. L'apparition des décodeurs satellites et des chaînes du câble, des consoles de jeux connectées et des box ADSL multi-services fait croître le nombre d'équipements connectés au domicile des utilisateurs. Ces équipements sont initialement configurés et développés pour une gamme précise d'applications et doivent être remplacés en cas d'évolution des technologies. La tendance générale est d'intégrer de plus en plus d'intelligence et de puissance de calcul dans ces équipements. Il est donc naturel d'initier un projet permettant l'exécution de composants logiciels téléchargés. Le framework OSGi repose sur un concept de Bundle qui est une archive java contenant des classes, des ressources et un descripteur d'archive. L'archive est installée sur une plate-forme d'exécution, le framework OSGi, qui se charge d'ouvrir le descripteur, d'identifier une classe de démarrage, de l'instancier automatiquement puis de lancer une fonction de démarrage. L'inversion de contrôle du framework OSGi est la plus simple possible et se résume à l'interface présentée dans l'algorithme 2.8 qui permet de "lancer" et "d'arrêter" un composant[32].

[28]`http://loom.codehaus.org/`
[29]`http://excalibur.apache.org/`
[30]`http://www.keelframework.org`
[31]Le projet Avalon reste cependant la meilleure expérience que j'ai pu observer sur le développement de logiciel libre. Ce projet était fortement tendu par des objectifs complexes, divergents et demandant des compétences techniques très pointues. Il reste un projet de référence dans son non-aboutissement et une excellente illustration de problèmes que l'ont peu observer dans la mise en œuvre de logiciel libre.
[32]Une réelle révolution par rapport à Java qui ne permet que de "lancer" une application par sa fonction main sans pouvoir "l'arrêter", puisque l'arrêt correspond à l'arrêt de la machine

Algorithm 2.8 Interface `org.osgi.framework.BundleActivator`

```
package org.osgi.framework;
public interface BundleActivator{
  public void start(BundleContext bc) throws Exception;
  public void stop(BundleContext bc) throws Exception;
}
```

Cette interface permet au conteneur de donner la main au composant bundle représenté par une instance de démarrage. Le conteneur transmet une référence, sur lui même afin que la classe puisse interagir avec lui. L'inversion du contrôle s'arrête là. La programmation orientée service proposée par le framework OSGi se fait par programmation. Un code peut enregistrer un service en s'adressant au `BundleContexte` transmis lors du démarrage du bundle. L'enregistrement d'un service Point est illustré dans l'algorithme 2.9 :

Algorithm 2.9 Enregistrement d'un service OSGi

```
package testserveur;
import org.osgi.framework.BundleActivator;
import org.osgi.framework.BundleContext;
import org.osgi.framework.ServiceRegistration;
public class PointImpl \\
       implements BundleActivator, Point {
  // Facette : Interface BundleActivator
  ServiceRegistration sr;
  public void start(BundleContext bc){
    ServiceRegistration sr=bc.registerService( \\
                "testserveur.Point", this, null);
  }
  public void stop(BundleContext bc){
    bc.unregister(sr);
    this.sr=null;
  }
  // Facette : Interface Point
  int x, y;
  public void deplace(int dx, int dy){
    this.x+=dx;
    this.y+=dy;
  }
}
```

Le *bundle* d'installation du service Point contient la classe `PointImpl` qui

virtuelle.

joue à la fois le rôle de classe d'implantation (interface Point) et de point d'entrée du *bundle* (interface `BundleActivator`) et éventuellement l'interface du service associé. La fonction d'inscription du service Point se fait dans l'appel `registerService`, qui prend 3 paramètres d'entrée. Le premier est l'interface associée au service, le deuxième est une implantation à disposition, et enfin le troisième (dans l'exemple `null`) est une liste de propriétés utilisable en cas d'implantations multiples d'un même service. Ces dernière propriétés sont fondamentales pour l'aspect dynamique car elles permettent de changer d'implantation en fonction de critères externes, comme un contexte par exemple. Ainsi on peut changer d'implantation lorsque le contexte change car une implantation différente peut alors être selectionnée. Dans cet exemple d'utilisation, on voit que le framework représenté par la référence sur le `BundleContext bc` joue le rôle d'un annuaire de services qui associe à une interface un certain nombre d'implantations de services. Il est important de noter qu'il n'y a aucune relation entre *bundles*, implantation de service et interface de service. Il est possible d'avoir un *bundle* qui amène une interface de service, mais que diverses implantations soient amenées par plusieurs *bundles* indépendants. Une implantation de service peut être remise en cause en cours d'exécution ce qui distingue fortement OSGi d'autres approches plus statiques comme Avalon par exemple.

Lorsqu'un *bundle* client désire manipuler un service il passe également par le pointeur du `BundleContext`. Par exemple, le code client de l'algorithme 2.10 permet d'interagir avec une des implantations disponibles de l'interface `Point`.

Algorithm 2.10 Manipulation d'un service OSGi

```
package test;
import org.osgi.framework.BundleActivator;
import org.osgi.framework.BundleContext;
import testserveur.Point;

public class Client implements BundleActivator{
  public void start(BundleContext bc){
    ServiceReference sr=bc.getServiceReference(Point);
    Point p=(Point)bc.getService(sr);
    p.deplace(50,50);
  }
  public void stop(BundleContext bc){
    //...
  }
}
```

On constate dans notre exemple que l'inversion de contrôle n'est imposée que pour la gestion du cycle de vie du bundle et reste trés simple, car l'objectif de l'intergiciel est de rester le plus générique possible et ne peut donc pas imposer un cycle de vie aussi complexe qu'Avalon par exemple.

Algorithm 2.11 Approche événementielle sur les services OSGi

```
package test;
import org.osgi.framework.BundleActivator;
import org.osgi.framework.BundleContext;
import org.osgi.framework.ServiceListener;
import org.osgi.framework.ServiceEvent;
import testserveur.Point;

public class PointImpl \\
      implements BundleActivator, ServiceListener{
  //Implantation de l'Interface BundleActivator
  public void start(BundleContext bc){
    bc.addServiceListener(this, \\
               "(objectClass=testserveur.Point");
  }
  public void stop(BundleContext bc){
  }
  //Implantation de l'Interface ServiceListener
  public void serviceChanged(ServiceEvent event){
    if (event.getType()==ServiceEvent.REGISTERED){
      ServiceReference sr=event.getServiceReference();
      Point service=(Point)bc.getService(sr);
      p.deplace(50,50);
    }
  }
}
```

D'autre part, lorsqu'un bundle client doit obtenir une référence à (se composer avec) une implantation de service il utilise un mécanisme de recherche contextuelle. Ce mécanisme permet d'obtenir une référence éphémère sur le service car elle n'est garantie que pour l'exécution de la fonction en cours d'exécution. Les implantations pouvant apparaître ou disparaître dynamiquement une référence sur une implantation ne peut pas être garantie en dehors de la fonction qui l'obtient. On tombe alors dans le même cas que celui des intergiciels distribués asynchrones. Aux mêmes causes, mêmes effets, on passe alors par un mécanisme réactif de notification du cycle de vie des services utilisés. Le framework OSGi fournit un mécanisme d'inversion de contrôle événementiel permettant à un client d'être notifié lorsqu'une implantation de service apparaît ou disparaît. Le code de l'algorithme 2.11 illustre ce principe de fonctionnement.

Dans cet exemple, le client s'abonne aux notifications liées à l'apparition et la disparition de services conformes à l'interface **testserveur.Point**. A chaque fois qu'un serveur enregistre un service, le code client est notifié par l'appel à la fonction `serviceChanged`. Le conteneur indique au client le type de modi-

fication et celui-ci réalise les actions en conséquence. Ce modèle de programmation événementiel est conforme à l'inversion de contrôle mais il est facile de constater qu'il est plus complexe à programmer qu'un modèle de programmation client/serveur. La programmation orientée service dans le framework OSGi est volontairement simple. Elle repose sur une interface minimale pour enregistrer ou obtenir une référence sur un service. Elle est cependant complète dans la mesure où elle permet d'avoir une approche réactive événementielle de la programmation. Il y a un certain nombre d'extensions au framework OSGi permettant de générer le code de liaison [33] automatiquement. Par exemple les projets comme *iPOJO [Escoffier et al. 2007]* où la section "declarative services" de la spécification R4 d'OSGi [Alliance 2007] permettent la génération automatique des dépendances entre services. Les services sont décrits dans des fichiers XML externes qui indiquent les dépendances existantes entre services. Des compilateurs se chargent de prendre en compte ces descriptions afin de générer du code permettant d'automatiser les interactions complexes entre les services.

OSGi propose une approche composant par la notion de *bundle*. Il permet de véhiculer des classes et des ressources entre un dépôt distant d'application et une machine virtuelle locale d'exécution par téléchargement sur TCP/IP. Le bundle est une archive qui contient les classes et les ressources et il permet d'isoler ces classes et ces ressources dans une machine virtuelle. Ce mécanisme se fait en associant à chaque *bundle* installé un chargeur de classe. Ce chargeur garantit que les classes fournies par un bundle n'interférerons pas avec des classes fournies par un autre si l'assembleur de composants ne les mets pas explicitement en relation. Ce mécanisme est fondamental dans l'installation de versions successives de classes, car il permet de mettre dans des *bundles* indépendants des versions différentes de la même classe. Le framework OSGi se charge de fournir à un client utilisateur la version désirée et garantit que deux versions de la même classe coexistent sans problème. La figure 2.3 illustre ce principe.

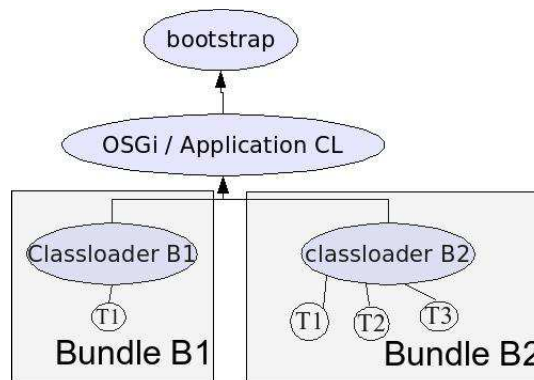

FIG. 2.3 – Organisation hiérarchique des chargeur de classes Java

[33]C'est à dire l'ensemble du code de glu que le programmeur doit ajouter pour mettre en relation deux services.

Les ovales représentent les chargeurs de classes. Le `bootstrap classloader` et `l'application classloaders` contiennent l'ensemble statique des classes mises à disposition dans la machine virtuelle : les classes apportées par les *bundles* B1 et B2 représentent la partie dynamique téléchargée. Il est tout à fait possible dans ce contexte que les classes T1 du *bundle* B1 et T1 du bundle B2 coexistent sans que cela ne pose un problème. Le *bundle* offre donc un concept d'isolation de classes grâce aux chargeurs de classes associés, et permettent donc également un retrait des classes de la machine virtuelle lorsqu'elles ne sont plus utilisées. Lorsqu'un *bundle* est retiré du framework, le *classloader* associé est détruit ainsi que toutes les classes qu'il a permis de charger en mémoire. Ce mécanisme de chargement/déchargement est fondamental pour avoir une longue durée de vie du système sans qu'il ne sature la mémoire[34].

Enfin, avant de charger une classe, un classloader doit demander à son parent s'il peut instancier la classe avant lui. Ce mécanisme empêche la surcharge des classes du système. Il est donc impossible en théorie de redéfinir une classe standard dans la mesure où elle est connue du bootstrap classloader. Cependant, dans le cadre de la programmation orientée service ou la programmation orientée composants, il est nécessaire de pouvoir instancier une classe provenant d'un bundle frère. OSGi met donc en place un mécanisme d'import/export de package Java permettant de mettre, dans une structure de "câblage", les packages échangés. Les importations et exportations de packages permettent l'instanciation de classes entre deux *bundles* qui serait sinon impossible du fait du mécanisme de délégation de chargement vertical.

OSGi présente donc un grand nombre d'intérêts pour le développement d'applications à base de composants et orientées services. Parmi les avantages on peut souligner les suivants :

- Il repose sur des mécanismes standards Java. Le framework OSGi fonctionne à la fois sur les anciens (jdk1.4) et les environnement contraints Java comme J2ME/CDC. La seule réelle contrainte est celle de posséder le mécanisme d'extension des chargeurs de classes, qui n'existe pas dans les environnements J2ME/CLDC par exemple.
- L'approche orientée composants permet d'ajouter de nouvelles classes en cours d'exécution. Ces nouvelles classes proviennent de sources potentiellement distantes et s'exécutent dans des chargeurs de classes isolés.
- L'approche orientée services permet de construire des applications à partir de briques de programmation élémentaires appelées services.

Le framework OSGi a permis de réunir deux mondes dans une même approche technologique. L'approche composant convient aux petits environnements, de bas niveau et contraints, car il est possible d'ajouter ou de supprimer de nouveaux composants sans réinitialisation du système et l'approche orientée services convient aux applications de haut niveau nécessitant une décomposition en service élémentaires. La figure 2.4 montre le double intérêt que l'on peut porter à OSGi. Les éléments sur la droite de la figure représentent des applications pos-

[34]C'est le point de différence majeur qui existe entre les *assemblies* de Microsoft .NET et les *bundles* OSGi

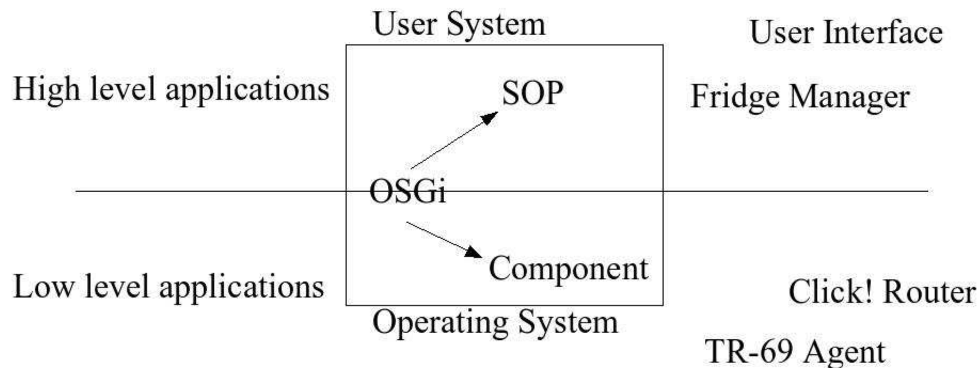

FIG. 2.4 – OSGi d'un point de vu applications système et applications utilisateur

sibles d'OSGi associées aux deux mondes. Interfaces utilisateur et Gestionnaire de refrigérateur pour des applications hautes et un routeur Click ! ou un agent de supervision TR69 pour les applications de bas niveau.

OSGi se retrouve aussi bien dans les noyaux d'exécution de plates-formes J2EE [BEA 2008][35], dans des projets de machines virtuelles Java [Apache Software Foundation 1998][36], que dans des applications dédiées aux utilisateurs finaux comme Eclipse [37].

Plus récemment (et sur la base des projets comme Avalon et OSGi) un certain nombre de projets ont émergé pour l'amélioration des principes de programmation orientée services. Les interrogations de ces projets portent sur les questions suivantes :

- Comment générer automatiquement et efficacement le code d'injection de services ?
- Comment exprimer facilement une application composée ?
- Comment intégrer des application existantes et comment ne pas dépendre d'un framework d'exécution ?

Il existe de très nombreux *frameworks* de support à la programmation orientée services et nous nous limiterons à citer ceux qui nous paraissent important. Dans le domaine de la programmation orientée service on pourra donc regarder SPRING [38], PICO [39] ET NANO CONTENEUR [40], GOOGLE GUICE [41] ET METRO [42]. Ces différentes plates-formes, à notre connaissance, sont focalisées sur le modèle de programmation orienté services et délaissent l'approche à composants pour le changement dynamique d'implantations. C'est la raison principale

[35] http://www.bea.com/framework.jsp?CNT=msa.jsp
[36] http://harmony.apache.org/
[37] http://www.eclipse.org/equinox/
[38] http://springframework.org/
[39] http://www.picocontainer.org/
[40] http://nanocontainer.codehaus.org/
[41] http://code.google.com/p/google-guice/
[42] http://www.dpml.net/

pour laquelle nous sommes restés sur la spécification OSGi pour nos implantations.

2.3 Une pile d'exécution standard pour environnements contraints et pervasifs

Par rapport à ce que nous avons présenté, la mise en œuvre d'une pile d'exécution pour les environnement contraints peut se faire de trois manières différentes. On peut la définir intégralement à partir de rien, dégrader un intergiciel applicatif pour ne conserver qu'un minimum de fonctionnalités, intégrer les services manquant dans un intergiciel système.

Définir intégralement une telle architecture pose un problème de compétence, de spécialisation et de main d'œuvre. Cette activité complexe est assimilée à la définition de nouveaux systèmes d'exploitation. Nous avons lancé des recherches qui descendent à ce niveau afin d'intégrer des approches à composants orientées services pour la gestion des couches réseau [Royon et al. 2005, Royon et al. 2004, Fournel 2004] mais ces projets sont restés marginaux.

Dégrader un intergiciel applicatif afin de retirer les services inutiles et d'ajouter des services spécifiques complémentaires est une approche que nous avons testé au début de nos activités [Al Masri et Frénot 2001, Al Masri et Frénot 2002b]. Le soucis est qu'on est alors tributaire des évolutions de ces intergiciels qui ne sont pas nécessairement compatibles avec notre volontée. De plus on est alors fortement lié à une implantation spécifique ainsi qu'à une spécification donnée comme J2EE. Enfin dans les intergiciels applicatifs, de nombreux apports sont liés exclusivement à la gestion des données (modèle relationnel, transactions...). Dans le cadre des environnements de nos projets, notre vision est plus focalisée sur les flux réseaux et les exécutions de calculs.

Enfin la dernière approche consiste à ajouter les services voulus à un intergiciel système et en regardant comment réaliser une integration simple et efficace. Convaincu des avantages apportés par les intergiciels systèmes pour la conception d'architectures logicielles complexe, nous nous sommes intéressés à l'intégration des services standards manquant au domaine d'exploitation visé. La pile de référence d'exécution standard est présentée figure 2.5.

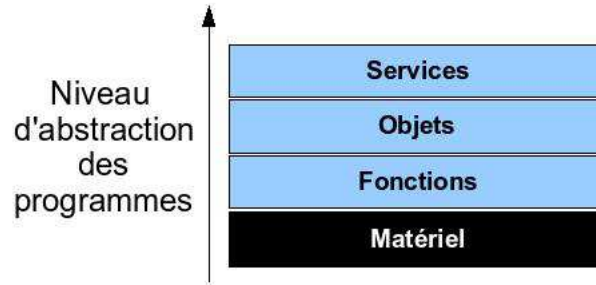

FIG. 2.5 – Pile d'abstraction des programmes

Nous nous sommes intéressés à l'implantation de cette pile dans le cadre d'une approche de passerelles matérielles domestiques dont la pile d'exploitation que nous mettons en œuvre est présentée dans la figure 2.6.

FIG. 2.6 – Pile d'exécution embarquée pour OSGi

En remontant la pile de bas en haut nous avons donc choisi :
- Un matériel de type ARM présentant des caractéristiques similaires à des environnements de passerelles domestiques (Linksys NLSU2 [NSLU2 2008]).
- Sur ce type de matériel il existe quelques systèmes d'exploitation disponibles. Après une étude et un ensemble de tests nous nous sommes focalisés sur le projet SlugOSBE[43]
- Au dessus de ce système d'exploitation nous utilisons la machine virtuelle JamVM[44] utilisée conjointement avec les classes standards du projet *gnu-classpath*[45]
- Au dessus de cette machine virtuelle nous utilisons la plate-forme Felix comme implantation d'OSGi.

Bien évidemment nous n'utilisons cette pile qu'à titre de validation de nos outils et de nos concepts. Elle nous permet de posséder un socle stable d'exécution et ainsi d'unifier et de standardiser nos tests et donc nos résultats. Nous réalisons

[43] `http://www.nslu2-linux.org/wiki/SlugOS/HomePage`
[44] `http://jamvm.sourceforge.net/`
[45] `http://www.gnu.org/software/classpath/classpath.html`

tous les outils sous des architectures classiques i386 puis nous les portons sur cette pile pour en vérifier le bon fonctionnement.

Si on observe cette pile d'un point de vue services, nous pouvons l'imaginer selon la figure 2.7.

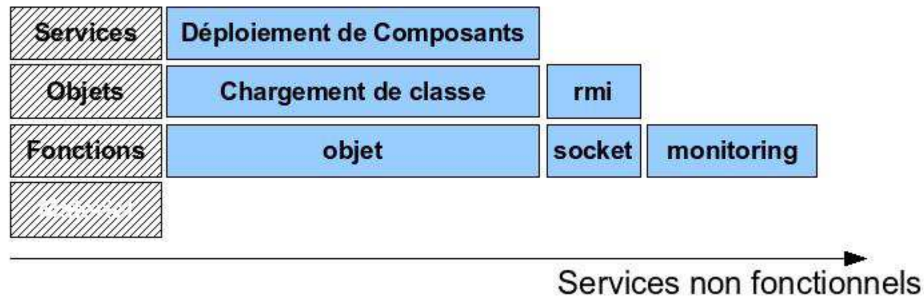

FIG. 2.7 – Services non-fonctionnels de la pile d'exécution utilisée

Sur cette figure nous avons représenté notre vision des architectures orientées services au moment du commencement de nos travaux. La pile offre un service de programmation orientée service représenté par la partie hachurée de gauche. La pile OSGi offre une approche orientée composants permettant de télécharger dynamiquement de nouvelles applications, mais en se déplaçant sur la droite de la figure nous voyons qu'il n'existe pas d'autre aspect associé à la gestion de services. Le reste de ce mémoire reflète notre activité de spécification et d'intégration des autres services qui nous sont apparus nécessaires à intégrer pour une exploitation d'un middleware système en environnement industriel domestiques et contraints.

Chapitre 3

Extensions aux intergiciels systèmes

Dans le second chapitre de ce document nous avons présenté la plate-forme OSGi et l'intérêt qu'elle suscite dans de nombreux contextes d'exploitation. Dans ce chapitre nous présentons des extensions à la passerelle OSGi afin d'offrir des fonctionnalités qui ne sont pas initialement proposées. Nous décrivons succinctement dans cette section les raisons qui nous ont amenées à définir tel ou tel aspect non fonctionnel, puis nous détaillerons la mise en œuvre sous OSGi de ces aspects.

Distribution OSGi permet de mettre en œuvre des applications, mais il n'est pas un intergiciel distribué, seule la récupération de composants peut se faire à distance. Nous nous sommes intéressés à la distribution de services en considérant OSGi comme un socle adéquat pour la réalisation d'applications à composants. L'idée de distribuer l'intergiciel système sur plusieurs nœuds de calcul a pour objectif de répartir les services entre ces différents nœuds et de réaliser une application distribuée dont on masque sa localité ou sa répartition. Nous avons appliqué cette approche dans des architectures de type grilles de calul.

Supervision L'intergiciel système se propose d'être un élément de convergence pour des environnements contraints comme les boitiers ADSL ou les téléphones mobiles. Ce type d'environnement doit être administré à distance et il est nécessaire d'intégrer de nombreuses couches permettant de remonter toutes les informations possibles dans les différents contextes d'exploitation possibles. Nous nous sommes penchés sur les principes de supervision associés afin de pouvoir gérer un grand parc de boîtiers ou garantir des performances suffisantes en terme de supervision. Cette activité représente une très grande partie du travail que nous avons mené au sein du CITI et dans le cadre du projet MUSE[1].

[1]Projet européen 6 FP : IST-6thFP-507295

Virtualisation La virtualisation est un principe fondamental permettant de fournir un espace de travail s'exécutant dans un autre espace de travail. L'espace virtualisé permet de faire croire à son utilisateur qu'il a un contrôle total des ressources qu'il utilise alors que le système ne lui en alloue qu'une partie. La virtualisation des systèmes est un défi fondamental pour les grands systèmes car elle permet un hébergement simultané de nombreuses applications, provenant éventuellement de fournisseurs concurentiels, sans se soucier des effets de bords. Notre cadre d'utilisation montre que pour les intergiciels systèmes, la virtualisation est tout autant un grand défi pour les systèmes à taille réduite que nous étudions.

Réduction des environnements d'exécution Les intergiciels systèmes permettent d'avoir une pile d'exécution très prometteuse. Pour l'instant l'exécution en environnement contraint reste une activité où l'approche classique consiste à mettre en place des solutions ad-hoc et/ou fortement dégradées. Nous présenterons nos approches liées aux systèmes embarqués qui reposent sur une pile d'exécution standard. Nous conservons ainsi un des principes de Java, qui est le Write Once Run Anywhere et qui sous-entend la portabilité des applications entre différents systèmes cibles quelles que soient leurs tailles.

Sécurisation La sécurité est le domaine non fonctionnel par essence. Tout système se doit d'être sécurisé. Dans le cas des passerelles dynamiques de services, les composants de services arrivent et disparaissent en cours d'exécution. Il est nécessaire d'observer les éléments de sécurité qui sont perturbés et de concevoir de nouvelles approches de sécurité pour les intergiciels systèmes. Nous entendons par domaines non fonctionnels des éléments comme l'identification, l'authentification, la signature, la privatisation des échanges, etc. Les outils de base comme la cryptologie et le chiffrement ne font pas partie de nos activités de recherche.

3.1 Organisation du chapitre

Nous présentons le détail des différents projets que nous avons menés autour de ces thématiques. La totalité des projets présentés gravitent autour de la pile Java/OSGi (sauf le premier qui concerne les plates-formes J2EE disponibles avant OSGi). Cette pile représente le seul socle d'exécution permettant de mettre en œuvre simplement nos extensions. Certains projets auraient pu être menés sur d'autres piles, notamment sur .NET, mais les coûts d'ingéniérie associés auraient été trop importants et cela n'aurait pas apporté plus d'informations.

Nous présentons les résultats de chacun de nos projets selon une structure similaire :

1. Justification et présentation de la problématique identifiée ;
2. Bibliographie complémentaire ;
3. Résultats et commentaires ;

3.2 Distribution pour OSGi

3.2.1 Pourquoi distribuer ?

Cette première approche que nous avons menée dans le cadre d'OSGi provient d'une généralisation que nous voulions faire des plates-formes J2EE. J2EE est une spécification dont les interactions envisagées entre les composants de haut-niveau (EJB, servlet....) sont par essence distribuées et réparties. Lorsque nous avons identifié qu'il était fondamental d'intégrer au cœur de ces systèmes une passerelle de services logiciels, notre travail s'est orienté selon 2 axes : un axe autour de la distribution de composants EJB sur une fédération distribuée de plates-formes J2EE qui s'est rapidement recentré sur la distribution de services sous OSGi.

Répartition d'EJB sur une fédération de serveurs Ce projet mené au laboratoire CITI de 2000 à 2004, vise à répartir le déploiement de composants logiciels EJB sur une fédération de serveurs. La fédération permet l'hébergement des différents composants. La figure 3.1 présente une chaîne de déploiement d'EJB dans le cadre d'une telle fédération.

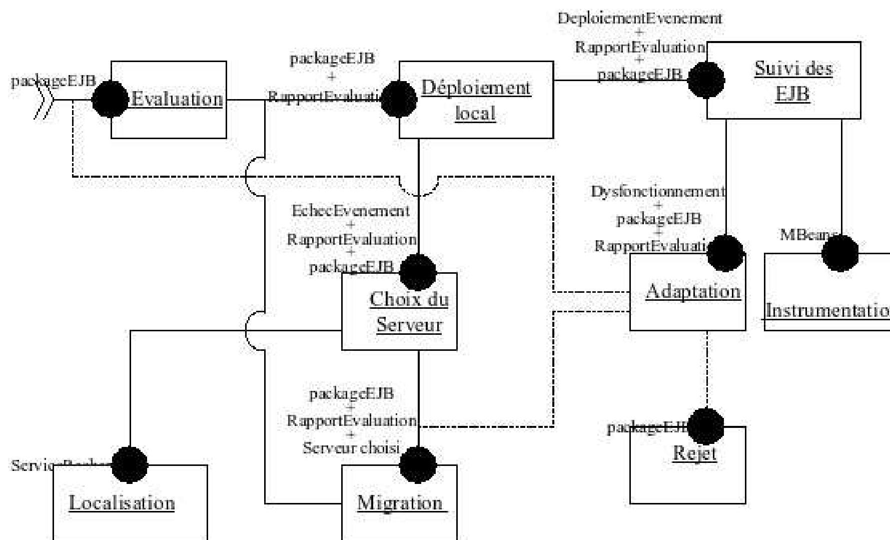

FIG. 3.1 – Chaîne de gestion d'EJB sur une fédération de serveurs

Le principe général est d'adjoindre à chaque EJB un descripteur de consommation de ressources afin qu'une "contractualisation" de l'exécution puisse se faire entre le composant et la plate-forme qui a été choisie pour son hébergement. Nous avons montré la relation existante entre un coût unitaire d'exécu-

tion d'un EJB avec un seul utilisateur distant et l'augmentation du coût en fonction du nombre d'utilisateurs distants simultanés. Une des problématiques est qu'il est impossible de faire "confiance" à un EJB. Si l'évaluation initiale sur un système off-line permet d'avoir une certaine approximation du coût en phase d'exploitation, il est quasiment impossible de garantir que tous les chemins d'exécution aient été testés. Il est donc nécessaire que les conteneurs de composants contrôlent, lors de l'exécution, les ressources effectivement consommées par les composants.

Ce projet a posé un problème majeur lié à la plate-forme cible utilisée. L'environnement J2EE est un environnement professionnel assez volumineux. L'exploitation dans le cadre d'une fédération nécessite d'intervenir sur de nombreuses lignes de codes qui étaient en constante évolution. D'autre part la plate-forme J2EE n'était pas réellement en adéquation avec les environnements contraints que nous envisagions comme cible d'étude au laboratoire. Nous nous sommes donc recentré sur le noyau d'exécution de ce type de plates-formes à savoir les passerelles de services et plus particulièrement le comportement d'une telle passerelle dans un environnement fédératif comme imaginé précédemment.

Distribution de services OSGi Le projet de distribution de services s'est alors recentré sur le cas d'OSGi. D'une part parce que nous percevions cette passerelle comme fondamentale pour être au centre de l'exécution d'architectures J2EE, et d'autre part parce qu'elle représentait une simplification réelle pour la mise en œuvre de notre fédération de services. Avoir un OSGi distribué nous permet de développer une application orientée service initialement locale et de pouvoir la répartir sur une fédération de nœuds de calculs sans en modifier le code. Ce principe d'exécution locale qui peut être répartie est ce que nous avons appelé une exécution prototypée localement. Pour réaliser cela, à chaque service fourni par un bundle nous générons automatiquement un bundle serveur RMI et un bundle client RMI permettant de dérouter automatiquement les appels entre un code demandeur et un service. L'interposition de l'appel RMI se fait en cours d'exécution par le framework OSGi et en prenant en compte la charge à l'instant de la requête de la machine. En cas de charge trop élevée le framework fabrique un bundle d'interposition client, recherche une passerelle moins chargée et redirige automatiquement l'appel sur celle-ci. L'idée fondamentale mise en œuvre est d'exploiter les demandes de liaison entre les composants client et fournisseur, imposées par la programmation orientée service. Au moment de cette demande d'association, nous générons automatiquement les appels de services distant équivalents.

Notre approche a pour objectif d'une part de permettre la répartition d'une application sans modification du code sur plusieurs nœuds de calcul pour l'appliquer par exemple dans le cadre d'une grille de calcul hétérogène et d'autre part de permettre le développement de plates-formes J2EE fédérées et distribuées sur les différents nœuds de calculs.

Nous avons appliqué cette architecture à deux applications de référence. Le premier exemple a été d'appliquer notre approche à une application de prédic-

tion de couverture wifi développée au sein du laboratoire. Nous avons convaincu l'équipe d'utiliser OSGi pour développer l'application et nous avons appliqué notre approche afin de montrer la potentialité de répartition de l'application sur plusieurs nœuds. L'application a été restructurée selon une approche MVC en séparant la configuration d'une simulation, la visualisation des résultats et les différents composants permettant le calcul en autant de services OSGi. Nous avons réparti ces services sur différents nœuds et nous avons montré que l'application était naturellement répartie par OSGi. Il est par exemple possible d'avoir un déport de l'affichage des résultats et de la saisie des paramètres de simulation sur l'écran d'un iPaq. OSGi réalise les invocations de services vers les nœuds de calculs performants situés sur des sites distants.

Nous avons également appliqué notre approche à une architecture répartie d'encodage de fichiers audio mp3. L'idée de l'application est d'avoir un code client s'exécutant, par exemple, dans une Applet java qui est capable de répartir les différentes piste d'un CD inséré, sur différentes machines afin qu'elles en réalisent un encodage mp3. L'application montre la répartition de la charge d'encodage entre les différents nœuds de la fédération.

3.2.2 Bibliographie complémentaire

Nous n'insisterons pas sur les plates-formes J2EE ou l'invocation de procédure distante de type RMI, mais nous recommandons la lecture de [SCK et Wyant 1994] qui positionne bien les problématiques associées. L'approche proposée pour étendre OSGi pour la distribution a été remise au goût du jour récemment dans les projets comme R-OSGi [Rellermeyer et al. 2007] de l'université de Zurich et certains travaux non publiés à ce jour de l'université de Lancaster. Ces deux approches refusent d'utiliser RMI comme couche de transport pour des raisons discutables de manque de performances, et on peut leur reprocher de ne pas être explicite sur les cas d'utilisation envisagés.

3.2.3 Résultats et Commentaires

Ces deux activités ont été déclenchées au lancement du laboratoire CITI. L'activité autour des passerelles J2EE a été faite dans le cadre de la thèse de Nada Al Masri [Masri 2005]. Nous avons également travaillé sur les évaluations de performance de RMI dans le cadre de J2EE [Frénot et Balan 2003, Brebner et al. 2005].

L'activité autour de la répartition des services a été réalisée dans le cadre de l'ACI DARTS[2], Déploiement et Administration de Ressources, Traitements et Services [Projet DARTS 2002b] [3] dont j'étais le pilote. Dan Stefan a été recruté en tant qu'ingénieur de recherche sur ce projet et a réalisé la majeure partie des implantations associées [Frénot et Stefan 2004b].

[2]Le document `http://tinyurl.com/4egzwc` donne une bonne synthèse des travaux réalisés dans le cadre du projet.

[3]`http://liris.cnrs.fr`

Distribuer des applications sur une fédération de machines s'est révélé relativement aisé. La difficulté majeure rencontrée est de trouver les cas d'utilisation significatifs démontrant l'intérêt de notre approche. Transformer une application locale en son équivalent distribué n'intéresse pas toujours le développeur de l'application car il est plus simple, dans le cas d'une application imaginée localement, d'acheter une plus grosse machine plutôt que de gérer la complexité liée à la répartition sur le réseau de calculs.

Qu'un serveur d'application J2EE Java puisse accéder aux quantités de ressources consommées par ses composants s'est révélé un des challenges les plus important de notre approche. La notion de supervision de nœuds de calcul s'est rapidement révélée comme étant un problème fondamental dans les projets que nous développions. Sans information simple sur les charges des différentes machines de la fédération il est impossible de répartir de manière intelligente les fonctions. Nous avons rapidement perçu ce problème et nous nous sommes réorientés sur les problématiques de supervision et de management d'applications OSGi.

3.3 Supervision pour OSGi

La supervision de passerelles OSGi a représenté la majeur partie de notre activité dans le laboratoire. Cette section est donc la plus importante. Les premiers résultats ont été exploré pour l'ACI DARTS et intégrés dans le cadre du projet européen MUSE [Projet MUSE 2007][4]. Une grande partie du code développé est intégré de manière officielle à la plate-forme open-source OSGi Felix de la fondation Apache [Apache Software Foundation 2005][5].

3.3.1 Pourquoi superviser ?

La supervision de systèmes couvre l'intégralité des activités autour d'un système qui ne sont pas l'exécution et l'ordonnancement des processus. Cette définition quelque peu provocatrice n'est pas très éloignée de la vérité. La recherche en système d'exploitation concerne très souvent l'optimisation des processus en exécution dans un espace mémoire fini et relègue à des activités annexes ce qui n'en relève pas. Cependant autour du cœur d'exécution des système d'exploitation il est nécessaire de standardiser et de mettre en œuvre de nombreuses activités qui se résument souvent à l'administration et à la supervision :

- `Monitoring` ou `supervision` : cette activité consiste à représenter l'état d'un système selon n'importe quelle métrique possible.
- L'**administration** repose sur la supervision et couvre 5 activités de base résumées par l'acronyme `FCAPS [FCAPS 2000] :`
  - `Fault management` : gestion du système en cas de panne,
  - `Configuration` : d'une manière générale tout ce qui est relatif à l'activité pré-exécution,

[4] `http://www.ist-muse.org`
[5] `http://felix.apache.org`

- `Accounting` : tout ce qui permet d'évaluer l'activité post-exécution,
- `Performance` : tout ce qui permet d'identifier et d'améliorer les performances du système,
- `Security` : activité transversale par essence.

L'information de supervision permet la prise de décision dans les couches d'administration afin d'orienter l'exécution du système dans une direction ou une autre.

La majorité des systèmes d'administration reposent sur une architecture à base de sondes qui remontent les informations de supervision dans une base de données. L'application d'administration exploite cette base de données pour réaliser ses propres actions.

Dans le projet précédent de distribution de calculs sur une grille, nous avons eu besoin de remonter des informations provenant des nœuds, actifs ou éteints, charge CPU courante ou encore quantité de mémoire restante. Cette information nous permettait de réaliser les prises de décision concernant le déploiement d'applications ou de remonter une alarme en cas d'inactivité d'un nœud. Dans notre première approche nous nous sommes basés sur l'outil Network Weather System[NWS 2003][6] (NWS) pour prédire les consommations. Nous avons intégré l'interaction avec NWS dans l'architecture de supervision JMX afin de pouvoir interagir avec NWS à partir de consoles de supervision Java.

Ce travail réalisé nous avons décidé de l'intégrer dans le framework OSGi afin de fournir au centre des passerelles de services logiciels un outil de supervision et d'administration.

Un tel outil permet :

- d'extraire les informations de supervision des passerelles,
- d'interagir avec une passerelle à partir d'une interface de supervision Java,
- de faciliter le développement de sondes de supervision.

Le projet `MOSGi` pour Managed OSGi est la mise en œuvre de cette plate-forme de supervision pour OSGi. Il est intégré depuis 5 ans dans la distribution standard Felix du projet Apache. Il a été développé selon une approche orientée service et repose sur un certain nombre de composants qui fournissent les différentes fonctionnalités du système :

- Le bundle Agent fournit l'interface de service pour inscrire ou retirer des sondes de supervision.
- Les bundles de connexion (HTTP et RMI) permettent d'interagir avec les sondes déployées à partir soit à partir d'un navigateur Web (connecteur HTTP), soit à partir d'une plate-forme de supervision ad-hoc par le protocole jmxremoting [JSR-160 2003].
- Chaque sonde est un *bundle* qui enregistre une ou plusieurs sondes de supervision auprès de l'agent.

La figure 3.2 illustre l'architecture générale de `MOSGi`.

Outre ces éléments classiques en supervision, nous avons ajouté une amélioration spécifique qui concerne l'intégration automatique de clients de supervision spécifiques. Dans les approches classiques, soit les plates-formes sont génériques

[6] `http://nws.cs.ucsb.edu/ewiki/`

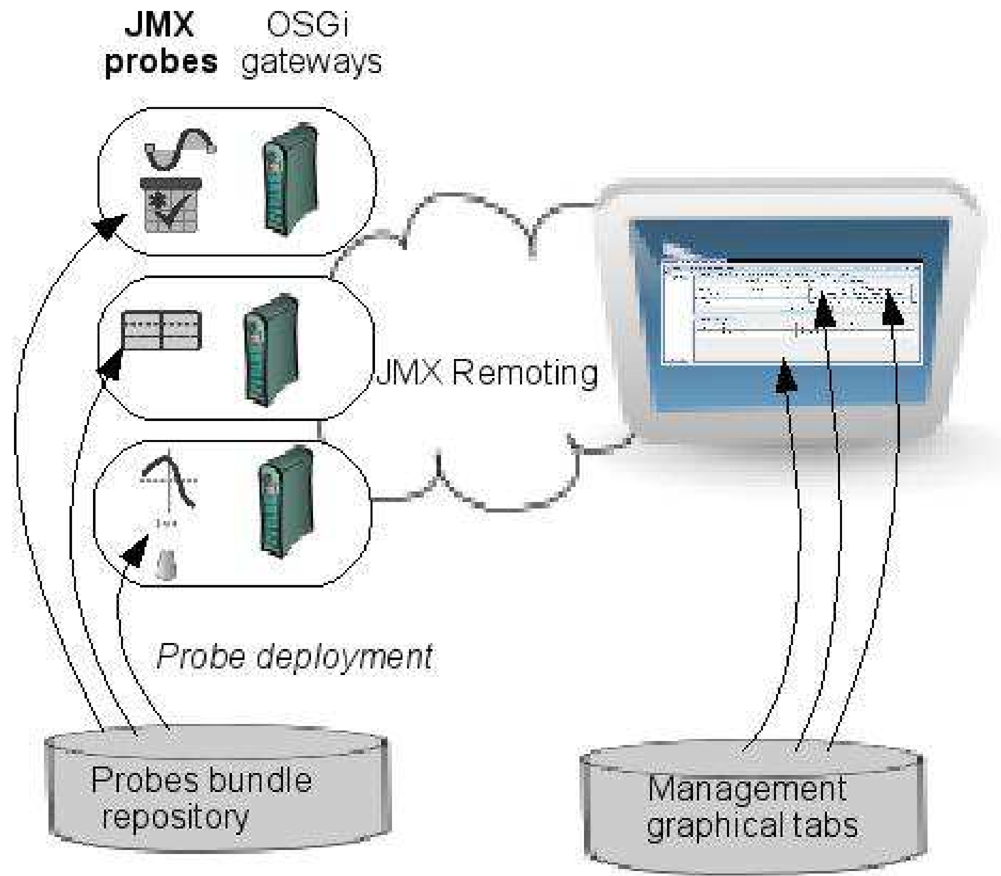

Fig. 3.2 – Architecture générale de MOSGi

et elles traitent toutes les informations de supervision à l'identique et génèrent donc des interfaces utilisateurs standards, soit elles sont spécifiques et doivent être configurées avant de pouvoir superviser les équipements spécifiques. Dans notre approche, nous exploitons le caractère dynamique d'OSGi pour récupérer à l'exécution les plugins de supervision des passerelles sur le client. Le client est une application OSGi qui, lorsqu'elle se connecte à un serveur OSGi à administrer, demande l'interface de supervision pour chaque sonde déployée. Pour chaque sonde, le client obtient une URL d'où il peut télécharger un *bundle* contenant l'interface d'interaction. Notre approche permet de fournir une plate-forme cliente de supervision initialement vide qui charge dynamiquement des modules de supervision en fonction des nœuds supervisés visités et des services qui y sont activés.

3.3.2 Bibliographie complémentaire

Il existe de nombreux systèmes de supervision d'applications dont on trouvera une synthèse dans [Olivier et André 2003]. La majorité d'entre eux repose sur le même principe d'agent, de base de données de supervision et de client d'administration. Nous avons choisi de nous fonder sur JMX [JSR-3 2000, JSR-160 2003], car ce protocole dédié aux applications Java, fournit quelques sondes standard (MXBeans) et s'intègre parfaitement à OSGi. Nous avons également étudié un certain nombre de systèmes de supervision comme SBLIM d'IBM[7] ou CIM-OM[8] fournis par le DMTF. Mais ces solutions reposent sur des librairies d'interrogation de bas niveau écrites en C et spécifiques à un environnement de type i386. Elles doivent donc être réecrites dans le cadre de passerelle matérielles de service sous processeur ARM.

3.3.3 Projets et Résultats associés

Comme nous l'avons souligné en introduction, l'administration de système est une vaste entreprise qui passe initialement par une architecture de supervision. Une fois cette architecture fournie nous avons travaillé sur deux thématiques de recherche liées. La première se focalise sur le déploiement de composants sur les passerelles et plus particulièrement sur un déploiement reposant sur un réseau pair-à-pair, la seconde se focalise sur l'évaluation du coût de supervision des passerelles de services.

Déploiement pair-à-pair de composants La plate-forme OSGi permet l'installation de composants à partir de dépôts de services. Dans une vision de type passerelle matérielle ADSL, il est nécessaire de déployer les nouveaux composants sur un grand nombre de passerelles simultanément. L'approche reposant sur un serveur central peut ne pas passer à l'échelle, à moins d'utiliser une infrastructure de déploiement dédiée. L'idée défendue dans notre cas

[7] `http://sblim.wiki.sourceforge.net`
[8] `http://www.openpegasus.org/`

[Frénot et Royon 2005] est d'utiliser les passerelles voisines comme sources d'installation des *bundles*. Lorsqu'une passerelle reçoit un nouveau *bundle* à installer, elle le met à disposition des autres passerelles pour une récupération par un protocole pair-à-pair. Nous avons réalisé un démonstrateur sous *FreePastry* [FreePastry 2004][9]. Nous avons développé un composant spécifique pour l'installation à travers le réseau *Pastry* et modifié la gestion des URL afin de pouvoir installer un bundle en émettant la commande `start p2p ://monBundle.jar`, qui intègre le schéma de recherche sur le réseau pair-à-pair. Notre architecture prend en compte les installations de versions de *bundles* et s'intègre à la plateforme de supervision pour connaître l'état du déploiement des composants.

Évaluation des charges de supervision induites sur des passerelles matérielles L'utilisation de MOSGi pour superviser des passerelles de services logicielles est simple à mettre en place. Dans le cadre du projet MUSE, nous nous situons dans un cadre concurrentiel où plusieurs fournisseurs peuvent proposer des services plus ou moins équivalents comme par exemple de la vidéo à la demande. Dans ce nouveau modèle économique, les fournisseurs de services sont indépendants du fournisseur de passerelle et du fournisseur de connectivité et doivent avoir des garanties sur la bonne ou la mauvaise exécution des services. Afin de pouvoir superviser le comportement de leurs services sur la passerelle domestique ils vont devoir interroger les sondes régulièrement. Dans une approche OSGi/Java dans un environnement de petite taille comme une passerelle domestique, ce coût de supervision n'est pas négligeable. Plus le client d'administration interroge la passerelle, plus il va générer une charge d'administration sur celle-ci. Au contraire, moins il demande d'information, moins il va générer de charge mais les informations récupérées ne seront peut être plus valides. La difficulté du problème est de trouver un équilibre entre la pertinence des informations récupérées et la charge induite sur la passerelle administrée. Évaluer le coût de supervision sur une passerelle Java/OSGi est complexe du fait des imprévisibilités de la machine virtuelle liées aux mécanismes additionnels comme le ramasse-miettes ou la compilation juste à temps. Nous avons fait la conception d'un système d'évaluation des coûts de supervision. Le coût de chaque sonde est évalué sur la machine cible et nous avons démontré que tant que la machine ne passe pas certaines limites de charge ce coût est directement proportionnel à la fréquence des requêtes d'interrogation. Le principe de l'outil de planification est de fournir à l'opérateur du service un moyen d'évaluer et de borner sa charge de supervision sur les passerelles. Par exemple, supposons qu'un opérateur de réfrigérateurs intelligents désire superviser ses équipements. Supposons alors que l'équipement possède une sonde remontant l'état du frigo (allumé / éteint), une sonde remontant la température intérieure, une autre indiquant la température extérieure et une dernière listant les provisions restantes. Si l'opérateur du réfrigérateur demande l'état de toutes les sondes toutes les 200ms, il génère une charge de 100% sur un petit équipement qui ne peut alors plus rien faire d'autre. L'outil proposé évalue la charge de chaque sonde et propose une

[9] `http://freepastry.rice.edu/FreePastry/`

planification des interrogations afin de lisser la charge et de n'obtenir que 5% de charge induite par la supervision de l'équipement qu'il supervise. Par exemple son plan de supervision interrogera la première sonde toutes les 30s, la seconde toutes les 10, la troisième toute les 1/2 heure et enfin la dernière une fois par jour. Chaque exploitant de service définit son plan de supervision et la charge de la passerelle est alors globalement anticipée. Une variation non prévue de la charge peut signifier, soit que le contrat de planification a été modifié, soit qu'un des composants ne fonctionne plus de manière conforme. Ce projet est décrit dans l'article [Frénot et al. 2008].

3.3.4 Résultats et Commentaires

La supervision et l'administration des passerelles de services est une pierre angulaire des architectures de livraison de services au domicile des utilisateurs. Étrangement c'est une chose très peu détaillée et spécifiée dans la documentation OSGi. Notre solution offre une excellente proposition du fait de sa souplesse et de sa simplicité de mise en œuvre.[10]

Nous ne connaissons pas le réel taux d'exploitation de l'implantation comme une `MOSGi`, mais nous recevons régulièrement des demandes d'amélioration de notre code, ce qui nous laisse penser que l'architecture est exploitée dans des cas d'utilisation industriels. L'architecture proposée n'est pas une architecture optimisée mais il s'agit d'une architecture de référence sur laquelle nous pouvons d'une part comparer d'autres approches et d'autre part proposer des évolutions dans de nombreuses directions.

Dans la première phase du projet MUSE, nous avons largement décrit et défendu cette architecture de supervision. Nous avons également suggéré un principe de multi-fournisseurs de services sur une passerelle. Si les industriels de MUSE voyaient le multi-fournisseur de manière statique où plusieurs fournisseurs sont en concurrence pour un même service comme la livraison de la télévision par exemple[11], nous nous sommes intéressés au cas où des fournisseurs ne sont initialement pas intégrés à la chaîne de services. Dans ce cas nous devons gérer le cycle de vie complet du système, à savoir : apparition/disparition d'équipement, apparation/disparition de fournisseur de services, apparition/disparition de logiciels associés.

3.4 Virtualisation pour OSGi

La notion de multi-fournisseur dans la cadre des passerelles de service logiciels est le fait de pouvoir changer de prestataire de service au fil du temps, sans

[10] Les industriels utilisent des approches spécifiques pour la supervision ADSL, comme le TR-69 du forum DSL [DSL Forum 2004], tr-69 `http://www.dslforum.org/techwork/tr/TR-069.pdf`.

[11] Dans cette approche le type de service étant initialement connu, les fournisseur s'entendent sur les fonctionnalités minimales attendues et la passerelle matérielle est optimisée en conséquence.

changer d'équipement. Notre étude sur OSGi/Java nous a menés à concevoir un système d'ajout de nouveaux prestataires de services au cours du temps.

3.4.1 Pourquoi virtualiser ?

Changer régulièrement de fournisseur de services permet de faire jouer la concurrence. Dans ce cas la difficulté est de pouvoir accueillir de nouveaux services en cours d'exécution. Cette fonctionnalité est au cœur du mécanisme Java/OSGi, mais ne peut pas satisfaire un exploitant de passerelles. Afin de pouvoir intégrer de nouveaux services il doit soit réaliser l'intégration pour garantir qu'un nouveau service ne perturbe pas les autres, soit fournir des garanties équivalentes si l'exécution n'est pas certifiée. Ce problème pourrait être similaire à ce que l'on a pu observer dans le domaine de la téléphonie mobile où il est possible d'installer de nouvelles applications dans un téléphone. La différence fondamentale est que dans notre approche les services sont exécutés en parallèle.

Une solution classiquement mise en œuvre pour permettre l'exécution simultanée d'applications ne devant pas interférer est de les isoler par virtualisation du système. Nous avons proposé et implanté une virtualisation d'OSGi. C'est à dire qu'il est possible d'exécuter une passerelle de services logiciels dans une autre passerelle de services. En exploitant notre architecture de supervision, un exploitant de services n'accède qu'au service qu'il doit voir et la virtualisation segmente ces différents services. La déclaration et le retrait d'une passerelle virtuelle de services se fait de manière simple par une interface d'administration de la passerelle racine.

Le cas d'utilisation envisagé est le suivant. Un utilisateur achète une passerelle de services matérielle sur étagère, comme un téléphone mobile avec ou sans abonnement. Son abonnement de base lui offre les services de connectivité, comme la télévision, internet et la voix. De plus, il peut acheter des équipement spécifiques, comme un réfrigérateur, un baladeur MP3 ou un disjoncteur électrique, dont les logiciels support d'administration sont automatiquement intégrés à sa passerelle. L'équipement se déclare à la passerelle racine et indique de quel fournisseur de services il dépend. Si le fournisseur associé possède un accord d'exécution, une passerelle virtuelle de services est démarrée sur la passerelle racine. Les services liés au nouvel équipement sont installés dans cette passerelle virtuelle et l'utilisateur peut soit administrer son équipement à partir de son accès internet, soit autoriser le fournisseur de l'équipement à réaliser des tâches d'administration comme le contrôle de l'état ou la surveillance de la consommation d'énergie.

Il est relativement facile d'isoler une exécution. A l'extrême il suffit de lancer une pile Java/OSGi dans un processus du système d'exploitation pour chaque fournisseur. Cette approche ne passe pas à l'échelle et ne permet pas de faire communiquer les services des passerelles entre eux. Nous avons réalisé cette isolation dans le cadre d'OSGi, mais nous avons intégré la possibilité pour une passerelle virtuelle d'accéder à des services soit d'autre passerelles virtuelles, soit à des services de la passerelle racine. Cette possibilité de récupérer les références de services permet l'échange et la récupération de services standards comme les

traces ou l'agent d'administration. L'architecture de passerelles virtuelles de services est représenté par la figure 3.3.

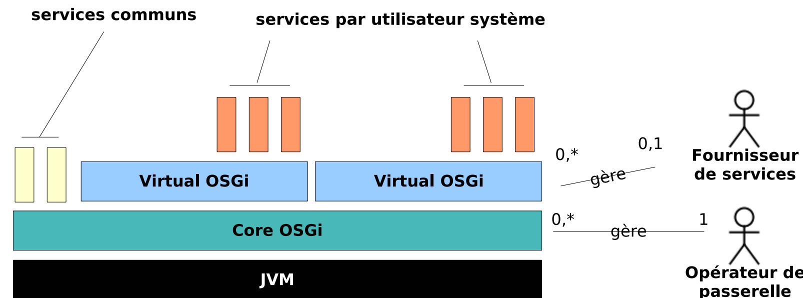

FIG. 3.3 – Organisation en passerelles virtuelles pour OSGi

3.4.2 Bibliographie complémentaire

La virtualisation de systèmes est surtout appliquée aux grands systèmes ou tout du moins aux systèmes présentant de hautes performances et partageables. La virtualisation [Smith et Nair 2005] peut se faire sur de nombreux plans. Les systèmes comme Xen [12][Barham et al. 2003], VMware [VMware 2006][13] ou VirtualBox [innotek GmbH 2008][14] sont largement utilisés. Dans le monde Java, les isolates [GC 2000] proposés sur Solaris pour les gros serveurs permet d'isoler l'exécution des grands calculs. La virtualisation dans le cadre d'OSGi et des petits équipements correspond parfaitement à un modèle économique que nous défendons et qui a été souligné dans [Franke et Robinson 2008]. L'apparition de nouvelles puces multi-cœurs chez ARM[15] pour environnements contraints nous conforte également sur la voie du multi-fournisseur par virtualisation. Cependant tout reste à définir dans ce contexte et notre approche nous semble prometteuse.

3.4.3 Résultats et commentaires

La virtualisation a été définie dans la seconde phase du projet MUSE pour répondre au modèle économique multi-services, multi-fournisseurs. Nous avons publié notre approche dans [Royon et al. 2006] et nous l'avons intégrée à la couche de supervision MOSGi. Une synthèse de ces deux activité est présentés dans [Royon et Frénot 2007b]. Yvan Royon a réalisé sa thèse [Royon 2007] dans le cadre de ces deux projets et son mémoire présente une très bonne vision de l'état de nos études. Il a également poursuivi notre réflexion dans le cadre d'une implantation spécifique C/Unix.

[12] `http://www.xensource.com/xen`
[13] `http://www.vmware.com/fr/`
[14] `http://www.virtualbox.org/`
[15] `http://tinyurl.com/MPCore`

Universal Gateway Model Universal Gateway Model est une étude menée avec Yvan Royon dans le cadre de sa thèse visant à comparer l'approche OSGi/Java intégrant nos concepts de virtualisation à une approche équivalente C/Unix. Le travail se focalise sur la vision composants et la résolution de dépendances entre composants natifs. Dans cette thèse nous avons clairement identifié les différents services fondamentaux qu'une passerelles de service s logiciels doit définir et fournir pour s'exécuter sur une passerelle de services matérielle. La figure 3.4 présente ces différents services :

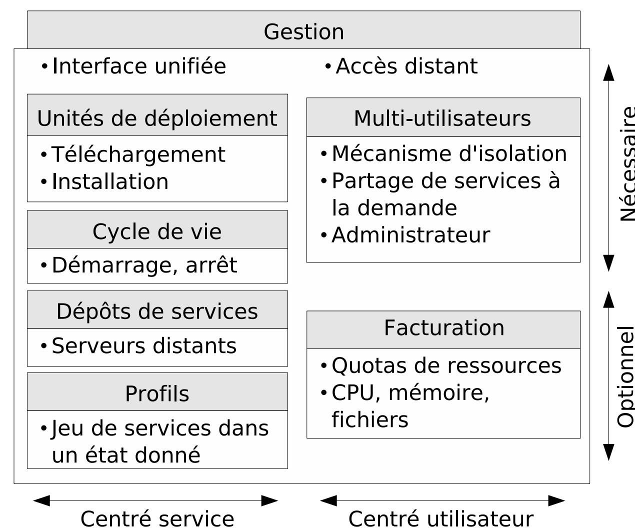

FIG. 3.4 – Architecture fonctionnelle de passerelles de services

– **Unités de déploiement :** concerne la récupération des composants et leur déclaration sur la passerelle ciblée.
– **Le cycle de vie** : concerne le lancement, l'arrêt, le démarrage des composants et/ou des services.
– **Les dépôts de services** : concernent la mise à disposition des composants sur des sites de référence.
– **Les jeux de profils** : améliorent les installations multiples de composants.
– **Le multi-utilisateur** : permet d'isoler l'exécution des services et l'installation des composants.
– **La facturation** : permet de controler finement et économiquement l'activité des services et des composants.
– **La gestion** : concerne l'accès générique et standard à tous ces services.

Ces différentes fonctions sont mis en œuvre d'une part avec la pile OSGi/Java et d'autre part avec une approche purement C/Unix. Home Gateway Linux (HGL) qui est l'implantation sous linux est limitée aux principes des composants sans aborder la programmation orientée services qui reste à définir pour le langage C. Cette architecture a été réalisée dans le cadre d'un partenariat avec Alcatel

Belgique Anvers, au cours d'un séjour qu'Yvan Royon a effectué dans leurs locaux.

Les projets MOSGi et VOSGi pour le management et la virtualisation d'OSGi présentent une architecture cohérente pour l'administration de services dans un contexte multi-fournisseurs. Cependant le contexte contraint des passerelles domestiques nous ont amenés à étudier spécifiquement des approches embarquées pour OSGi.

3.5 OSGi enfoui

OSGi est initialement conçu pour s'exécuter sur des environnements de petite taille comme des assistants personnels ou des passerelles domestiques. Cette possibilité n'est actuellement pas envisagée par les fournisseurs de tels équipements, mais nous pensons que la pression du multi-fournisseur amènera les opérateurs de passerelles à basculer vers des environnements standards et ouverts comme la pile OSGi/Java. Dans le cadre du projet MUSE II nous avons cherché à faire prendre conscience à nos partenaires industriels Alcatel, Thomson et ST-MicroElectronics qu'une machine virtuelle Java pouvait s'exécuter sur des environnements limités en mémoire et en puissance de traitement comme on peut les trouver dans les passerelles domestiques actuelles.

3.5.1 Pourquoi embarquer OSGi ?

OSGi a été valorisé comme une infrastructure facile à intégrer dans des environnements contraints. Cette vérité n'est pas si simple dans la réalité des implantations. Actuellement les contraintes imposées par Java font qu'il est relativement difficile d'intégrer cette pile de services dans l'architecture visée. Les caractéristiques typiques sont par exemples celles des équipements NSLU2 de Linksys [NSLU2 2008][16] :

- 8Mo de disque flash,
- 32Mo de mémoire SDRAM,
- Un processeur ARM9 à 266Mhz.

Elles sont quasiment similaires sur des équipements de type iPaq d'HP, NLSU2 de LinkSys ou des modem ADSL commercialisés par Thomson.

En observant les caractéristiques de ces systèmes il apparaît qu'ils possèdent beaucoup plus de mémoire que de disque, alors que dans les systèmes plus classique c'est l'inverse. Nous avons cherché à exploiter cette différence afin d'optimiser l'exécution de notre pile OSGi/Java. L'idée fondamentale que nous démontrons est d'exploiter à sa juste valeur les capacités d'extensibilité de la machine virtuelle Java en démarrant un environnement minimaliste stocké sur le disque flash puis en chargeant rapidement les classes restantes à partir de sites distants. L'approche orientée composants d'OSGi nous facilite grandement cette approche en identifiant clairement les classes à télécharger, et en les regroupant

[16] http://www.nslu2-linux.org/wiki/Info/CPUOverview

dans des *bundles*. Cette approche nous permet de garder une des justifications de Java qui est le "*write once, run anywhere*".

Nous avons défini l'architecture ROCS [Frénot et Papastefanos 2008] pour Remote OSGi Cache Server dont l'objectif est de réaliser un déport du chargement des classes sur un serveur RMI distant. L'idée est de gérer toute l'information de description des *bundles* installés sur la passerelle de manière classique, mais que l'intégralité des classes soit stockée sur les disques de machines serveurs. Lorsqu'un *bundle* doit fournir une classe, son *byte-code* est téléchargé par RMI sous la forme d'un tableau d'octets et transmis au chargeur de classes pour la définition de la classe. Dans notre approche, seule la description des *bundles* installés ou déployés réside sur la passerelle. L'intégralité du *byte-code* est à distance et n'est téléchargée, qu'au fur et à mesure des besoins des applications.

Cette approche n'est pas suffisante car la machine virtuelle arrive avec environ 40MB de classes standards. Ces classes standards ne peuvent donc pas être stockées sur la passerelle. L'ensemble des classes de la bibliothèque standard se décompose en 3 catégories par rapport au démarrage de la pile Java/OSGi :

1. les packages qui n'interviennent pas dans le processus de démarrage ;
2. les classes qui n'interviennent pas dans le processus de démarrage mais qui sont dans un package dont certaines classes interviennent dans le processus de démarrage ;
3. les classes qui interviennent dans le processus de démarrage.[17]

Le cas numéro 1 est géré par un *bundle* qui contient et exporte tous les packages. Les cas 2 et 3 sont à regrouper car l'implantation de la politique de sécurité de Java interdit de charger des classes d'un même package à partir d'origines différentes. Cet ensemble représente un volume de 2,5MB. En modifiant l'implantation de la politique de sécurité il est possible de déporter les classes du cas 2 ce qui réduit l'ensemble strict des classes nécessaires au démarrage à 860KB.

Notre approche réduit donc le volume de classes installées sur la passerelle à 2,5MB et réalise le chargement des nouvelles classes à partir d'un serveur RMI distant. Notre approche nous permet de faire tourner un environnement OSGi/Java sur un périphérique contraint en support de stockage local. En terme de performance notre approche apporte deux modifications :

- Le chargement de *bundle* n'est plus à la charge de la passerelle, mais uniquement du serveur RMI. La passerelle ne récupère que les informations d'état et de version du *bundle* installé. Dans nos tests nous gagnons 5s sur les 20s de démarrage de la passerelle.
- Le chargement d'une classe à partir d'un serveur RMI induit un surcoût proche de 50% du temps de chargement local sur la mémoire flash. Dans nos tests nous perdons 5s sur le chargement des 300 classes nécessaires au démarrage de la pile applicative (cas 3 présenté précédemment).

Il apparaît que globalement les temps de réponses en architecture locale sont les mêmes que sous l'architecture Rocs. Nous avons cependant réduit l'espace

[17] Pour connaître ces trois ensembles il suffit de démarrer une passerelle OSGi sur une passerelle matérielle de services et d'observer les classes chargées. Nous maintenons une bibliothèque d'outils pour extraire ces informations sur le site de la forge INRIA [INRIA 2008].

de stockage local à la passerelle de 9MB en utilisant le GNU-CLASSPATH à 2,5MB. Il est également à noter que sur certains équipements l'accès à la mémoire flash est extrêmement lent et que nos performances sont encore meilleures. Enfin nos exemples supposent un accès RMI à un serveur distant suffisamment disponible. Notre proposition est que ce serveur soit implanté sur les nœuds de diffusion ADSL au niveau des DSLAM locaux. La connexion RMI est dans ce cas au maximum de sa performance mais reste sous le contrôle de l'opérateur de services.

3.5.2 Bibliographie complémentaire

Les piles d'exécution réduites Java existent depuis les débuts de Java. On les retrouve dans les environnements de type J2ME [J2ME 2008],[18] avec les deux profils CDC et CLDC, respectivement pour les environnements de type téléphone portable et pour les environnements de type carte à puce. Cependant ces environnement, utilisent des machines virtuelles Java dédiées et développées de manière spécifique. Nous nous sommes focalisés sur les machines virtuelles open-sources afin de pouvoir facilement les faire évoluer. Wikipedia maintient une bonne liste de référence de machines virtuelles [VM List 2008][19]. Afin de réduire leur taille, certaines machines virtuelles open-source comme Mika [Mika 2008][20] redéfinissent les classes standards. Notre approche a été de conserver une machine virtuelle la plus proche du standard J2SE en exploitant de manière intelligente les librairies standards associées. JamVM [JamVM 2008][21] est un interpréteur de byte-code Java qui repose sur l'implantation open-source classpath [GC 2008][22] qui correspond parfaitement à nos besoins.

3.5.3 Projets et Résultats associés

Ces travaux sont les plus récents dans l'activité de notre équipe. Ils sont très prometteurs car ils permettent enfin d'exploiter des machines virtuelles sur de petits environnements connectés. Les travaux ont été spécifiés et réalisés dans le cadre d'une étude du projet MUSE2 sur les machines virtuelles contraintes [Frénot et Papastefanos 2008]. Les évaluations et les implantations réalisées montrent que l'architecture est tout à fait envisageable. De plus il n'existe pas de dépendance forte, envers RMI par exemple. Les classes récupérées à partir des serveurs distants peuvent l'être via n'importe quel protocole de transport de données. Nous avons choisi RMI pour des soucis de simplicité et d'efficacité de mise en œuvre.

Les sections précédentes ont détaillées nos activités non fonctionnelles autour d'OSGi (administration, virtualisation, enfouissement). Un dernier axe non fonctionnel sur la sécurité nous est rapidement apparu comme fondamental.

[18] http://en.wikipedia.org/wiki/Java_Platform
[19] http://en.wikipedia.org/wiki/List_of_Java_virtual_machines
[20] http://www.k-embedded-java.com/mika/trac/
[21] http://jamvm.sourceforge.net/
[22] http://www.gnu.org/software/classpath/

3.6 Sécurisation d'OSGi

Nous avons identifié cette thématique dés nos premiers travaux sur OSGi et nous avons choisi de proposer un sujet de thèse autour des problématiques associées. Durant sa thèse, Pierre Parrend a étudié et a proposé de nombreuses approches pour adjoindre des niveaux de sécurité tolérables à OSGi. La mise en place de la sécurité soulève un certain nombre de questions qui ne sont pas que techniques et la tâche liée à la mise en place de la sécurité dans ce type d'infrastructure n'est pas simple.

3.6.1 Pourquoi sécuriser ?

Nous n'allons bien évidemment pas soulever cette question qui semble évidente et largement débattue dans de nombreux ouvrages. Cependant la question de savoir quoi sécuriser est fondamentale. Nous avons choisi d'avoir une approche *top-down*, c'est à dire que nous avons d'abord choisi de mettre en place une architecture de sécurisation globale pour, petit à petit, resserrer la sécurité sur des points spécifiques plus précis. Nous avons proposé des modèles de sécurité sur trois plans : l'architecture générale de déploiement, la passerelle OSGi et les composants.

Architecture générale L'architecture générale d'OSGi permet l'installation de composants logiciels à partir de sites distants. L'approche standard de sécurisation passe par un principe de signature des composants qui leur permet d'être acceptés sur la plate-forme. Le contrôle de validité de signature d'OSGi repose sur celui défini par Java pour le contrôle d'archives jar avec quelques extensions spécifiques. Nous avons constaté qu'il n'existait pas d'implantation de la spécification proposée par OSGi et nous avons donc développé une telle implantation en *open-source*. Celle-ci propose un outil de gestion des signatures, et l'intégration dans le déploiement des composants OSGi d'un nouvel état de rejet d'un composant dont la signature n'est plus valide. Nous avons utilisé les approches standards de signatures à clé publiques et nous exploitons des bibliothèques en code libre pour la réalisation. Une fois l'architecture globale de sécurité validée nous nous sommes recentrés sur la passerelle elle-même pour analyser les problèmes de garantie d'exécution.

Garanties d'exécution de la passerelle Lorsqu'un *bundle* est signé, il peut être installé sur la passerelle. Cependant il n'est pas garanti que ce *bundle* ait un comportement sain. Dans VOSGi le principe est que plusieurs fournisseurs de services installent de manière autonome leurs propres *bundles*. Même si les différents fournisseurs sont clairement identifiés, il n'est pas garanti que les *bundles* installés fonctionnent sainement. Notre approche a été d'étudier l'intégralité de la spécification de la passerelle afin d'identifier les différents biais de sécurité. A la suite de cette étude nous avons implanté un catalogue de composants exploitant ces vulnérabilités. Ce catalogue de composants sert alors de plan de test et d'évaluation des différentes plates-formes OSGi. Nous avons réalisé des tests

de déploiement de ces composants sur les différentes implantations *open-source* d'OSGi disponibles, Felix [Apache Software Foundation 2005][23], Concierge[24] [Rellermeyer et Alonso 2007a] et Knopferlish [Gatespace Telematics 2006][25], afin d'en démontrer certaines défaillances lors d'une utilisation malicieuse. Nous avons ensuite proposé une implantation "durcie" d'OSGi, à partir de la souche Felix, permettant de circonscrire 20% des composants malicieux (7 failles sur 32 tests) si on ne modifie pas la machine virtuelle de support. L'objectif du catalogue et de la passerelle OSGi "durcie" est de définir des règles pour la sécurisation d'OSGi. Cependant la majorité de ces règles de peuvent pas être intégrés à la spécification OSGi, car elles couvrent principalement des problèmes d'implantation et non pas des problèmes de spécification de l'architecture. Nous avons donc démontré dans notre approche que le respect de la spécification n'est pas suffisant pour garantir une implantation sécurisé des passerelles OSGi.

Garanties d'exécution des composants Lorsque l'on veut de la sécurité dans Java, la méthode classique est de valider le lancement du gestionnaire de sécurité Java. Ceci se fait en lançant la machine virtuelle avec la commande suivante : `java -Djava.security.manager.` A partir de cet instant, les méthodes exécutées sont contrôlées par rapport à un fichier d'expression des droits. Par exemple, lorsque la méthode permettant l'ouverture d'une `Socket` est invoquée et si le gestionnaire de sécurité est activé, le contrôleur de sécurité peut contrôler d'une part la localisation de l'exécution de la méthode et d'autre part éventuellement la légitimité du signataire. Cette approche pose un problème de performance car lors d'un appel de fonction qui doit contrôler les droits, toutes les fonctions dans la pile d'appel doivent être contrôlées également. L'approche reposant sur le securityManager standard Java est assez coûteuse en temps en ajoutant un surcoût d'exécution d'environ 30%. Si ce surcoût peut être absorbé sur des gros systèmes en ajoutant de la puissance de calcul, c'est plus difficilement envisageable sur des environnements de passerelles domestiques, car la machine virtuelle Java représente déjà un sur-coût d'exécution par rapport à un code natif.

Cependant, OSGi offre une ouverture intéressante que nous avons exploitée. En effet, l'installation d'un *bundle* passe par une phase de déploiement sur la passerelle. Cette phase permet de contrôler la validité de l'archive aussi bien en terme de droits qu'en terme de dépendances requises. Il est possible d'imaginer une analyse statique du code pour vérifier les droits sur les méthodes utilisées. Cette phase de déploiement est fondamentale car la rupture existant entre installation et exécution permet d'effectuer un maximum de contrôles a priori pour avoir une exécution garantie. Notre projet Component Based Access Control (CBAC) [Parrend et Frénot 2008b] propose une analyse du code des *bundles* au moment du déploiement qui contrôle l'ensemble des appels réalisés. Si les appels concernent des fonctions "sensibles" de la machine virtuelle, les

[23]`http://felix.apache.org/site/index.html`
[24]`http://concierge.sourceforge.net/`
[25]`http://www.knopflerfish.org/`

droits sont contrôlés et le *bundle* est autorisé ou non à s'installer. CBAC présente de nombreux avantages par rapport à l'approche classique. Par exemple il est très facile d'ajouter de nouvelles fonctions considérées comme "sensibles" car leur liste est définie dans un fichier de configuration et non pas directement dans les codes des classes. De plus les fonctions "sensibles" peuvent également inclure des fonctions apportées par les *bundles* nouvellement installés.

Le lecteur intéressé par l'approche de sécurisation globale du déploiement de services se référera à [Parrend et Frénot 2006b, Parrend et Frénot 2007b], ainsi qu'aux livrables publics de MUSE [Festor et D'Haeseleer 2006]. La littérature autour de la sécurité est abondante, mais il n'en reste pas moins que de nombreuses implantations ne sont pas d'une clarté limpide dans les détails de leur mise en application. Nous concernant, les codes développés dans le cadre de cette activité sont disponibles sur le site de l'INRIA `http://sfelix.gforge.inria.fr`.

3.6.2 Projets et Résultats associés

Nous avons développé cette activité dans le cadre de la seconde phase du projet MUSE. A la suite des résultats prometteurs de notre approche nous participons à l'ANR LISE (Liability Issues in Software Environment) [Projet LISE 2010][26], débutée en janvier 2008, qui consiste à étudier les problèmes de confiance et de droit juridique autour des traces d'exécution de systèmes. Notre travail dans ce projet consiste à proposer une infrastructure de traces sécurisées pour OSGi. Les traces ne concernent que les activités autour des services. En réduisant la quantité de trace générée nous supposons qu'il sera largement plus simple d'analyser et d'identifier les causes de défaillance.

3.7 Conclusion

Le cœur de nos travaux a été présenté dans ce chapitre. Nous proposons un ensemble d'extensions non-fonctionnelles aux plates-formes de services logicielles existantes. Les trois axes principaux des travaux couvrent les communautés du management, des systèmes d'exploitation et de la sécurité. Ces trois communautés sont couvertes par l'intermédiaire d'un outil de convergence unique à savoir la plate-forme de services OSGi. Si cette démarche est très "appliquée" elle correspond d'une part à la vision de l'INSA de Lyon qui forme des ingénieurs "appliqués" et d'autre part à notre soucis de transversalité dans notre activité de recherche.

La reconnaissance de nos travaux s'est fait dans l'intégration d'une partie de nos codes dans le projet Apache Felix. De nombreux utilisateurs exploitent les fonctionnalités offerte par la couche de supervision, et nous recevons de plus en plus de sollicitations de la part de PME qui souhaitent intégrer ce type d'architecture dans des boîtiers pour superviser différents types d'environnements.

[26] `http://www-anr-ci.cea.fr/scripts/home/publigen/content/templates/show.asp?P=145&L=FR&ITEMID=17`

Nos travaux ont également été reconnus dans le cadre des travaux autour du projet MUSE. Car bien que nous n'étions pas initialement dans le cœur de la problématique de MUSE (livraison de services ADSL), nous avons, dès la première phase, identifié les prochains challenges industriels dans ce domaine. S'il était absolument impossible en phase I du projet de parler de machines virtuelles dans les set-tops box, le discours a complètement basculé en 2006, au milieu de la seconde phase. Nous avons donc parfaitement atteint nos objectifs de reconnaissance de nos activités et ceci, à la fois dans le domaine académique par nos publications, et industriel par nos contacts et les contrats associés.

Le second point que nous soulignons concerne les potentiels d'extension de ce type de plate-forme. Nous avons produit chaque année 2 à 3 sujets d'étude permettant d'accueillir des étudiants en stages, en projet de fin d'études ou encore sur des projets à plus long terme pour des ingénieurs de recherche. Ce type de système est au cœur du fonctionnement d'un grand nombre de systèmes diversifiés. En effet les plates-formes de type OSGi servent aussi bien comme cœur d'exécution de plates-formes de développement comme Eclipse que dans des environnement d'*e-business* très complexes comme la plate-forme J2EE. Il est alors très facile de comprendre qu'il est nécessaire de standardiser et d'implanter de nombreux services non-fonctionnels standards (comme la géolocalisation ou la gestion d'énergie), et que tous ces nouveaux services devront être facilement intégrés dans ces architectures.

L'intégration de services non-fonctionnels peut se faire sur toutes les couches de notre pile d'exécution pour en optimiser le fonctionnement. Par exemple, sur le système d'exploitation hôte pour en améliorer les performances, sur la machine virtuelle pour y intégrer certaines des fonctionnalités dès les plus bas-niveaux (comme les annuaires de services), et enfin dans les couches OSGi pour en améliorer les modèles de programmation. Toutes ces améliorations correspondent également à des communautés spécifiques de recherche pour l'amélioration des environnement d'exécution : système, administration, sécurité.

En 2004 nous avons recruté un second maître de conférences dans l'équipe. Frédéric Le-Mouël a démarré une activité autour de l'intégration de services et particulièrement autour de l'intégration contextuelle de services. Dans cette approche nous nous sommes intéressés à deux aspects non-fonctionnels. D'une part à l'intégration de composants dont les interfaces sont naturellement incompatibles et d'autres part à l'installation prenant en compte le contexte d'exécution pour des composants offrant les mêmes fonctionnalités. Pour l'adaptation de services on trouvera plus d'information dans [Ibrahim et al. 2007b, Le Mouël et al. 2006, Ibrahim et al. 2007a]. Pour l'adaptation contextuelle de services les articles qui y font référence sont [Ben Hamida et al. 2008c, Ben Hamida et al. 2008b, Ben Hamida et al. 2007]. Ces travaux ont été réalisés dans le cadre du projet Européen Amigo [27][Projet Amigo 2007]

Enfin, le dernier point à souligner est que notre activité de recherche se positionne dans une niche de recherche. Elle se place sur diverses frontières, n'étant ni du système, ni de l'administration, ni de la sécurité mais tout à la fois.

[27]`http://www.hitech-projects.com/euprojects/amigo`

Le positionnement de nos intergiciels systèmes se veut résolument comme une "tête de pont" entre systèmes, réalisant un rôle de passerelle entre divers univers et possédant en filigrane des problématiques d'intégration et d'optimisation.

Le dernier chapitre de ce document donne les grandes pistes que nous envisageons pour la suite de nos travaux.

Chapitre 4

Conclusion

Notre activité de recherche depuis 10 ans s'est focalisée sur l'ajout de services non fonctionnels à des supports d'éxecution orientés composants. Chaque service correspond à une compétence spécifique à un domaine de recherche et nous avons toujours cherché à valoriser notre travail dans la communauté correspondante.

Ainsi l'axe sur la distribution a été valorisé dans le domaine de la grille et des systèmes répartis, la supervision dans le domaine du management, la virtualisation dans le domaine du système d'exploitation et l'embarqué dans celui du middleware. Il est assez difficile de trouver une seule communauté à cibler pour notre activité, car cette approche transversale est en contradiction avec une vision pointue d'un domaine. Il est également difficile de se faire reconnaître dans une compétence unique puisque nous avons maîtrisé et exploité des approches technologiques aussi variées que celles issues de la supervision, de la communication, de la programmation, des systèmes pair-à-pairs ou encore des architectures à base de processeurs ARM. Les débuts de notre activité de recherche après thèse ont coïncidé avec la création du laboratoire CITI. Comme membre fondateur, j'ai défendu l'idée d'aller vers des activités de recherche transversales aux différents domaines de l'informatique, des réseaux et des télécommunications. Dans notre sphère de travail, comme - je l'espère - le démontre ce mémoire, nous avons été un des moteurs de cette tendance et nous avons démontré son potentiel.

L'ensemble de nos travaux nous a amené à côtoyer les communautés du management, des systèmes d'exploitation, des approches à composants, de la sécurité et des systèmes pairs-à-pairs. Cette diversité est très intéressante car elle nous permet de présenter à ces différentes communautés des approches différentes. Cela montre cependant un certain manque de communication et de dialogue entre ces communautés qui se spécialisent trop et deviennent fortement hermétiques à la convergence des concepts. Il est de plus difficile d'être présent aux différentes manifestations nationales et internationales des trois domaines. Cela fait que nos travaux sont à la frange des diverses communautés car souvent perçus comme un effort principal d'intégration. Mais notre expérience nous montre que l'intégration est également une activité de recherche qui présente des intérêts importants. Par exemple, en plus des résultats directs de

nos études, nos travaux apportent des cas d'utilisation concrêts, des éléments de comparaison ou encore des suites de tests réapplicables dans d'autres environnements. Notre sentiment est que la notion d'intégration de concepts et de systèmes a été laissée aux industriels dont le rôle est de prendre les technologies à la pointe dans diverses communautés et d'en réaliser la convergence. Pourtant, une reflexion plus théorique en accélèrerait le processus et permettrait l'émergence de nouvelles problématiques. Nous pensons que l'intégration est une réelle direction de recherche à exploiter qui soulève de nombreuses questions, comme : l'intégration est-elle toujours possible ? Quelle énergie de développement/conception faut-il fournir pour intégrer tel ou tel domaine ? Comment garantir une intégration complète ?

Nos prochaines directions de recherche

Nous sommes convaincus que l'informatique de 2010 se doit d'être une informatique de convergence, en proposant des outils et des architectures permettant d'intégrer les besoins issus de différents domaines et d'en tirer le meilleur parti. Notre activité d'intégration portera essentiellement sur les petits équipements, très souvent nomades, parfois mobiles, qui peupleront les architectures ambiantes pour capturer les informations provenant des capteurs alentours. Ils auront des caractéristiques similaires aux architectures "contraintes" sans être conçus de manière spécifique à ces usages. Ils seront les plus autonomes possible, et serviront pratiquement tout le temps d'équipements de médiation entre deux univers. C'est le cas des "Box" aux domiciles, des têtes de réseaux de capteurs sans fil ou encore des assistants personnels gérant un « body area network ».

Ces équipements présentent des contraintes de ressources classiques comme la puissance de traitement ou de stockage mais également de nouvelles contraintes comme la consommation énergétique ou l'exploitation en milieu hostile. Au-dessus de ces contraintes matérielles et contextuelles, ils doivent héberger un middleware ambiant qui permet d'une part de gérer les très petits équipements et d'autre part de fournir les données et les services aux utilisateurs ou aux systèmes globaux de supervision. Un tel middleware ambiant doit intégrer les éléments décrits dans ces travaux. La suite de nos travaux doit dans un premier temps viser à optimiser ce travail d'intégration. Nous pensons pouvoir améliorer cette intégration selon plusieurs pistes. Le génie logiciel et la génération de code, la formalisation des systèmes, la modélisation des interactions sont les outils de base nécessaires à une meilleure mise en oeuvre de middlewares ambiants.

Un autre élément fondamental de notre vision de l'avenir est de proposer des systèmes de programmation de plus en plus adaptés et focalisés. La tendance des langages de programmation comme Java est de proposer des outils de précompilation et d'automatisation d'écriture de code (Annotations [1], Proxy

[1] `http://java.sun.com/j2se/1.5.0/docs/guide/language/annotations.html`

dynamiques [2], Génération dynamique de byte Code [3], Aspect [4]). Nous pensons qu'il est nécessaire de passer à une étape d'identification et de développement de nouveaux langages permettant une intégration plus homogène des fonctionnalités. Dans ce contexte, l'objectif est de fournir tout ce qui permettra à un programmeur non expert de la programmation de définir de nouvelles architectures par intégration automatique des différentes fonctionnalités souhaitées.

Nous pensons que l'aspect simulation et formalisation sont deux briques qui manquent encore dans nos systèmes. Dans notre domaine particulier de travail, les démonstrateurs sont absolument nécessaires pour la validation des approches. Le monde de l'intelligence ambiante nécessite des démonstrateurs à grande échelle, très difficilement déployables pour les uniques besoins de la recherche. Dès lors, trois pistes s'offrent à nous : greffer nos concepts sur des environnements réels en production (type équipements publicitaires interactifs, stations Velov'), faire de la simulation à l'échelle des composants logiciels, ou mieux, faire de l'émulation de ces services sur un réseau, certes expérimental, mais bien réel.

La mise en place d'un middleware ambiant passe le plus souvent par une approche de sur-couche à un système d'exploitation existant. Nous pensons qu'une piste importante est d'arrêter l'isolation franche existant actuellement entre système d'exploitation, machine virtuelle et middleware. L'objectif est de remettre en question l'approche en couche des systèmes, afin de trouver une approche plus fluide de type nuage de services internes et externes, où l'information, les services et les systèmes sont partout et doivent toujours être intégrables et exploitables sans interruption des services fournis.

[2] `http://java.sun.com/j2se/1.3/docs/guide/reflection/proxy.html`
[3] `http://asm.objectweb.org/`
[4] `http://en.wikipedia.org/wiki/Aspect-oriented_programming`

Annexe A

Webographie

[Apache Cocoon 2005] Apache Cocoon, *The Cocoon CMS*, 2005
http://cocoon.apache.org/. 2.2.3

[Apache JServ 1995] Apache JServ, *The JServ Web Server*, 1995
http://archive.apache.org/dist/java/jserv/. 2.2.3

[Apache Software Foundation 1995] Apache Software Foundation, *The Avalon Project*, 1995
http://avalon.apache.org. 2.2.3

[Apache Software Foundation 1998] Apache Software Foundation, *Harmony : Open Source Java SE*, 1998
http://harmony.apache.org/. 2.2.3

[Apache Software Foundation 2005] Apache Software Foundation, *Felix OSGi R4 Service Platform implementation*, since 2005
http://felix.apache.org/. 3.3, 3.6.1

[Ares 2002] Ares, *Architecture et RésEaux de Services*, since 2002
http://www.citi.insa-lyon.fr/team/ares/. 1.1

[BEA 2008] BEA, *microService Architecture (mSA)*, 2008
http://www.bea.com/framework.jsp?CNT=msa.jsp\&FP=/content. 2.2.3

[Citi 2000] Citi, *CenTre d'Intégration et d'Innovation de services*, since 2000
http://www.citi.insa-lyon.fr. 1.1

[DSL Forum 2004] DSL Forum, *TR-069 : CPE WAN Management Protocol*, May 2004
http://www.dslforum.org/techwork/tr/TR-069.pdf. 10

[Excalibur 2005] Excalibur, *a lightweight, embeddable Inversion of Control container*, 2005
http://excalibur.apache.org/. 2.2.3

[Fowler 2004] Martin Fowler, *Inversion of Control Containers and the Dependency Injection pattern*, 2004
http://martinfowler.com/articles/injection.html. 2.2.2, 2.2.3

[FreePastry 2004] FreePastry, *Rice University, Houston, Texas*, 2004
http://freepastry.rice.edu. 3.3.3

[Gatespace Telematics 2006] Gatespace Telematics, *Knopflerfish OSGi framework*, 2006
http://www.knopflerfish.org/. 3.6.1

[GC 2008] GNU-Classpath, *GNU Classpath*, 2008
http://www.gnu.org/software/classpath/. 3.5.2

[gui 2008] *Guice*, 2006–2008
http://code.google.com/p/google-guice/. 2.2.3

[IETF-1050 1988] IETF-1050, *RPC : Remote Procedure Call protocol specification*, April 1988
http://tools.ietf.org/html/rfc1050. 2.1.1

[IETF-1094 1989] IETF-1094, *NFS : Network File System protocol Specification*, March 1989
http://tools.ietf.org/html/rfc1094. 2.1.1

[IETF-1832 1995] IETF-1832, *XDR : External Data Representation standard*, March 1995
http://tools.ietf.org/html/rfc1832.html. 2.1

[innotek GmbH 2008] innotek GmbH, *The VirtualBox project*, since 2008
http//www.virtualbox.org/. 3.4.2

[INRIA 2008] INRIA, *Gforge*, 2008
http://gforge.inria.fr/. 17

[J2ME 2008] Wikipedia J2ME, *Java Platform, Micro Edition*, 2008
http://en.wikipedia.org/wiki/Java_Platform%2C_Micro_Edition. 3.5.2

[JamVM 2008] JamVM, *JamVM virtual Machine*, 2008
http://jamvm.sourceforge.net/. 3.5.2

[JSR-160 2003] JSR-160, *Java Management eXtensions Remote API*, October 2003
http://www.jcp.org/en/jsr/detail?id=160. 3.3.1, 3.3.2

[JSR-3 2000] JSR-3, *Java Management eXtensions Specification*, 2000
http://www.jcp.org/en/jsr/detail?id=3. 2.2.3, 2.2.3, 3.3.2

[Keel 2005] Keel, *Keel : Meta Framework*, 2005
http://www.keelframework.org/. 2.2.3

[Loom 2005] Loom, *an extensible Component Container*, 2005
http://loom.codehaus.org/. 2.2.3

[Mika 2008] Mika, *Embedded Java Solutions*, 2008
http://www.k-embedded-java.com/mika/trac/. 3.5.2

[nan 2008] *Nano Container*, 2003–2008
http://nanocontainer.codehaus.org/. 2.2.3

[NSLU2 2008] NSLU2, *The NSLU2 CPU Overview*, since 2008
http//www.nslu2-linux.org/wiki/Info/CPUOverview. 2.3, 3.5.1

[NWS 2003] NWS, *Network Weather Service*, October 2003
http://nws.cs.ucsb.edu/ewiki/. 3.3.1

[OSGi Alliance 2002] OSGi Alliance, *The OSGi Alliance web site*, 2002
http://www.osgi.org/. 2.2.3, 2.2.3

[pic 2007] *Pico Container*, 2003–2007
http://www.picocontainer.org/. 2.2.3

[Projet Amigo 2007] Projet Amigo, *Ambient intelligence for the networked home environment, IST-004182 Framework Programme 6*, 2004-2007
http://www.hitech-projects.com/euprojects/amigo/. 3.7

[Projet DARTS 2002] Projet DARTS, *Déploiement et Administration de Ressources, Traitements et Services*, 2002
http://darts.insa-lyon.fr/index.html.en. 1.2, 3.2.3

[Projet LISE 2010] Projet LISE, *Liability Issues in Software Environment*, 2008–2010
http://www-anr-ci.cea.fr/scripts/home/publigen/content/templates/show.asp?P=145&L=FR&ITEMID=17/. 1.2, 3.6.2

[Projet MUSE 2007] Projet MUSE, *MultiServices EveryWhere, IST-026442 Framework Programme 6*, 2004–2007
http://www.ist-muse.org/. 1.2, 3.3

[Specification 1998a] Sun Specification, *Java DataBases Connectivity*, 1998
http://java.sun.com/javase/technologies/database/. 2.2.2

[Specification 1998b] Sun Specification, *Java Messaging Service*, 1998
http://java.sun.com/products/jms/. 2.2.2

[Specification 1998c] Sun Specification, *Java Naming and Directory Interface*, 1998
http://java.sun.com/products/jndi/docs.html#12. 2.2.2

[Specification 1998d] Sun Specification, *Java Transaction API*, 1998
http://java.sun.com/javaee/technologies/jta/index.jsp. 2.2.2

[VM List 2008] Wikipedia VM List, *List of Java virtual machines*, 2008
http://en.wikipedia.org/wiki/List_of_Java_virtual_machines. 3.5.2

[VMware 2006] VMware, *The VMware project*, since 2006
http//www.vmware.com. 3.4.2

Annexe B

Bibliographie

[Al Masri et Frénot 2001] NADA AL MASRI et STÉPHANE FRÉNOT, « Speech Recognition Integration in Medical Information System », *MedInfo*, London, England, October 2001. 2.3

[Al Masri et Frénot 2002b] NADA AL MASRI et STÉPHANE FRÉNOT, « Instrumentation dynamique d'applications à base d'EJB », *Journées Composants*, Grenoble, France, October 2002. 2.3

[Alliance 2007] OSGI ALLIANCE, *OSGi Service Platform, Core Specification, Release 4, Version 4.1*, OSGi Alliance, 2007, 4.1 édition. 2.2.3

[Apache Cocoon 2005] APACHE COCOON, *The Cocoon CMS*, 2005 `http://cocoon.apache.org/`. 2.2.3

[Apache JServ 1995] APACHE JSERV, *The JServ Web Server*, 1995 `http://archive.apache.org/dist/java/jserv/`. 2.2.3

[Apache Software Foundation 1995] APACHE SOFTWARE FOUNDATION, *The Avalon Project*, 1995 `http://avalon.apache.org`. 2.2.3

[Apache Software Foundation 1998] APACHE SOFTWARE FOUNDATION, *Harmony : Open Source Java SE*, 1998 `http://harmony.apache.org/`. 2.2.3

[Apache Software Foundation 2005] APACHE SOFTWARE FOUNDATION, *Felix OSGi R4 Service Platform implementation*, since 2005 `http://felix.apache.org/`. 3.3, 3.6.1

[Ares 2002] ARES, *Architecture et RésEaux de Services*, since 2002 `http://www.citi.insa-lyon.fr/team/ares/`. 1.1

[Arthur et Azadegan 2005] JOHN ARTHUR et SHIVA AZADEGAN, « Spring Framework for Rapid Open Source J2EE Web Application Development : A Case Study », *Sixth International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing and First ACIS International Workshop on Self-Assembling Wireless Networks*

(SNPD/SAWN'05), p. 90–95, IEEE Computer Society, Los Alamitos, CA, USA, 2005. 2.2.3

[Barham et al. 2003] Paul Barham, Boris Dragovic, Keir Fraser, Steve Hand, Tim Harris, Alex Ho, Rolf Neugebauer, Ian Pratt et Andrew Warfield, « Xen and the Art of Virtualization », *SOSP*, New York, USA, October 2003. 3.4.2

[Bates et Basham 2008] Bert Bates et Bryan Basham, *Head First Servlets and JSP : Passing the Sun Certified Web Component Developer Exam*, O'Reilly Media, 2008. 2.2.1

[BEA 2008] BEA, *microService Architecture (mSA)*, 2008
http://www.bea.com/framework.jsp?CNT=msa.jsp\&FP=/content. 2.2.3

[Ben Hamida et al. 2007] Amira Ben Hamida, Frédéric Le Mouël, Stéphane Frénot et Mohamed Ben Ahmed, « Approche pour un chargement contextuel de services sur des dispositifs contraints », *6ème atelier sur les Objets, Composants et Modèles dans l'ingénierie des Systèmes d'Information (OCM-SI'2007) at INFORSID*, Perros-Guirec, France, May 2007, http://www.le-mouel.net/Research/Publications/Workshops/2007/OCM-SI2007/OCM-SI2007.pdf. 3.7

[Ben Hamida et al. 2008b] Amira Ben Hamida, Frédéric Le Mouël, Mohamed Ben Ahmed et Stéphane Frénot, « Chargement Contextuel de services par coloration de graphes de dépendances », *Notere*, Lyon, 23-27 June 2008. 3.7

[Ben Hamida et al. 2008c] Amira Ben Hamida, Frédéric Le Mouël, Stéphane Frénot et Mohamed Ben Ahmed, « Une approche pour un chargement contextuel de services dans les environnements pervasifs », *Networking and Information Systems / Ingénierie des Systèmes d'Information (ISI)*, p. 59–84, vol. 13, n°3, juin 2008, Special edition. 3.7

[Brebner et al. 2005] Paul Brebner, Emmanuel Cecchet, Julie Marguerite, Petr Truma, Octavian Ciuhandu, Bruno Dufour, Lieven Eeckhout, Stéphane Frénot, Arvind S Krishna, John Murphy et Clark Verbrugge, « Middleware Benchmarking : Approaches, Results, Experiences », *Concurrency Computat. : Pract. Exper*, p. 1799–1805, vol. 17, 2005. 3.2.3

[Citi 2000] Citi, *CenTre d'Intégration et d'Innovation de services*, since 2000
http://www.citi.insa-lyon.fr. 1.1

[DSL Forum 2004] DSL Forum, *TR-069 : CPE WAN Management Protocol*, May 2004
http://www.dslforum.org/techwork/tr/TR-069.pdf. 10

[Eddon et Eddon 1997] Guy Eddon et Henry Eddon, *Inside Distributed COM*, Microsoft Pr, 1997. 2.1.1

[Escoffier et al. 2007] Clément Escoffier, Richard S. Hall et Philippe Lalanda, « iPOJO : an Extensible Service-Oriented Component Framework », *SCC 2007. IEEE International Conference on Services Computing*, p. 474–481, July 2007. 2.2.3

[Excalibur 2005] EXCALIBUR, *a lightweight, embeddable Inversion of Control container*, 2005
`http://excalibur.apache.org/`. 2.2.3

[FCAPS 2000] FCAPS, *Telecommunication standardization sector of ITU*, February 2000
TMN management functions - M.3400 - `http://www.itu.int/rec/T-REC-M.3400/en`. 3.3.1

[Festor et D'Haeseleer 2006] OLIVIER FESTOR et SAM D'HAESELEER, *Specification of Residential Gateway configuration*, MUSE IST-6thFP-026442, public deliverable, DB4.3, november 2006. 3.6.1

[Fournel 2004] NICOLAS FOURNEL, *Approche par composants pour une machine virtuelle micro-noyau*, Rapport de DEA INSA Lyon, 2004. 2.3

[Fowler 2004] MARTIN FOWLER, *Inversion of Control Containers and the Dependency Injection pattern*, 2004
`http://martinfowler.com/articles/injection.html`. 2.2.2, 2.2.3

[Franke et Robinson 2008] CARSTEN FRANKE et PHILIP ROBINSON, « Autonomic Provisioning of Hosted Applications with Level of Isolation Terms », *Fifth IEEE Workshop on Engineering of Autonomic and Autonomous Systems EASE*, p. 131–142, March 31-April 4 2008. 3.4.2

[FreePastry 2004] FREEPASTRY, *Rice University, Houston, Texas*, 2004
`http://freepastry.rice.edu`. 3.3.3

[Frénot et al. 2008] STÉPHANE FRÉNOT, YVAN ROYON, PIERRE PARREND et DENIS BERAS, « Monitoring Scheduling for Home Gateways », *IEEE/IFIP Network Operations and Management Symposium (NOMS 2008)*, Salvador Bahia, Brazil, April 2008. 3.3.3

[Frénot et Balan 2003] STÉPHANE FRÉNOT et TUDOR BALAN, « A CPU Resource Consumption Prediction Mechanism for EJB deployment on a federation of servers », *Workshop on Benchmarking at OOPSLA*, Anaheim, CA, USA, 2003. 3.2.3

[Frénot et Papastefanos 2008] STÉPHANE FRÉNOT et SERAFEIM PAPASTEFANOS, *Virtual Machines for Embedded Environments*, MUSE IST-6thFP-026442, public deliverable, DB3.6, january 2008. 3.5.1, 3.5.3

[Frénot et Royon 2005] STÉPHANE FRÉNOT et YVAN ROYON, « Component Deployment Using a Peer-To-Peer Overlay », *Working Conference on Component Deployment*, Grenoble, France, 28-29 November 2005. 3.3.3

[Frénot et Stefan 2004b] STÉPHANE FRÉNOT et DAN STEFAN, « M-OSGi : Une plate-forme répartie de services », *Notere*, Saïdia, Maroc, jun 2004. 3.2.3

[Gatespace Telematics 2006] GATESPACE TELEMATICS, *Knopflerfish OSGi framework*, 2006
`http://www.knopflerfish.org/`. 3.6.1

[GC 2000] GRZEGORZ CZAJKOWSKI, « Application Isolation in the Java Virtual Machine », *ACM SIGPLAN Conferences on Object-Oriented Programming,*

Systems, Languages, and Applications (OOPSLA), Minneapolis, MN, 2000. 3.4.2

[GC 2008] GNU-Classpath, *GNU Classpath*, 2008
`http://www.gnu.org/software/classpath/`. 3.5.2

[Goldman Rohm 1998] Wendy Goldman Rohm, *L'affaire Microsoft*, First Edition, 1998. 2

[Goodman 1995] Danny Goodman, *Complete Hypercard 2.0 Handbook*, Bantam Books, 1995. 2.2.1

[gui 2008] *Guice*, 2006–2008
`http://code.google.com/p/google-guice/`. 2.2.3

[Halvorson 2008] Michael Halvorson, *Microsoft Visual Basic 2008 Step by Step*, Microsoft Press, 2008. 2.2.1

[Hilyard et Teilhet 2008] Jay Hilyard et Stephen Teilhet, *C# 3.0 Cookbook*, O'Reilly Media, 2008. 2.2.1

[Ibrahim et al. 2007a] Noha Ibrahim, Fréderic Le Mouël et Stéphane Frénot, « C-ANIS : A Contextual, Automatic and Dynamic Service-Oriented Integration Framework », *Proceedings of the International Symposium on Ubiquitous Computing Systems (UCS'2007)*, *LNCS*, vol. 4836, p. 118–133, Springer Verlag, Tokyo, Japan, novembre 2007. 3.7

[Ibrahim et al. 2007b] Noha Ibrahim, Frédéric Le Mouël et Stéphane Frénot, « Automatic Service-Integration Framework for Ubiquitous Environments », *Proceedings of the International Conference on Mobile Ubiquitous Computing, Systems, Services and Technologies (UBICOMM'2007)*, Papeete, French Polynesia (Tahiti), France, November 2007. 3.7

[IETF-1050 1988] IETF-1050, *RPC : Remote Procedure Call protocol specification*, April 1988
`http://tools.ietf.org/html/rfc1050`. 2.1.1

[IETF-1094 1989] IETF-1094, *NFS : Network File System protocol Specification*, March 1989
`http://tools.ietf.org/html/rfc1094`. 2.1.1

[IETF-1832 1995] IETF-1832, *XDR : External Data Representation standard*, March 1995
`http://tools.ietf.org/html/rfc1832.html`. 2.1

[innotek GmbH 2008] innotek GmbH, *The VirtualBox project*, since 2008
`http//www.virtualbox.org/`. 3.4.2

[INRIA 2008] INRIA, *Gforge*, 2008
`http://gforge.inria.fr/`. 17

[J2ME 2008] Wikipedia J2ME, *Java Platform, Micro Edition*, 2008
`http://en.wikipedia.org/wiki/Java_Platform%2C_Micro_Edition`. 3.5.2

[JamVM 2008] JamVM, *JamVM virtual Machine*, 2008
`http://jamvm.sourceforge.net/`. 3.5.2

[JSR-160 2003] JSR-160, *Java Management eXtensions Remote API*, October 2003
http://www.jcp.org/en/jsr/detail?id=160. 3.3.1, 3.3.2

[JSR-3 2000] JSR-3, *Java Management eXtensions Specification*, 2000
http://www.jcp.org/en/jsr/detail?id=3. 2.2.3, 2.2.3, 3.3.2

[Keel 2005] KEEL, *Keel : Meta Framework*, 2005
http://www.keelframework.org/. 2.2.3

[Kernighan et Ritchie 2004] BRIAN W. KERNIGHAN et DENNIS M. RITCHIE, *Le langage C Norme ANSI, 2nd édition*, Dunod, 2004. 2.2.1

[Le Mouël et al. 2006] FRÉDÉRIC LE MOUËL, NOHA IBRAHIM, YVAN ROYON et STÉPHANE FRÉNOT, « Semantic Deployment of Services in Pervasive Environments », *RSPSI workhop at Pervasive 2006*, dublin, Ireland, 7-10 may 2006. 3.7

[Loom 2005] LOOM, *an extensible Component Container*, 2005
http://loom.codehaus.org/. 2.2.3

[Lutz 2007] MARK LUTZ, *Learning Python*, O'Reilly Media, 2007. 2.2.1

[Masri 2005] NADA AL MASRI, *Modèle d'Administration des Systèmes Distribués à Base de Composants*, Thèse de Master, INSA Lyon, 2005. 1.2, 3.2.3

[Mika 2008] MIKA, *Embedded Java Solutions*, 2008
http://www.k-embedded-java.com/mika/trac/. 3.5.2

[Moock 2007] COLIN MOOCK, *Essential ActionScript 3.0*, Adobe Developer Library, 2007. 2.2.1

[nan 2008] *Nano Container*, 2003–2008
http://nanocontainer.codehaus.org/. 2.2.3

[Niemeyer et Peck 1998] PAT NIEMEYER et JOSH PECK, *Exploring Java, 2nd edition*, Oreilly & Associates Inc, 1998. 2.2.1

[NSLU2 2008] NSLU2, *The NSLU2 CPU Overview*, since 2008
http//www.nslu2-linux.org/wiki/Info/CPUOverview. 2.3, 3.5.1

[NWS 2003] NWS, *Network Weather Service*, October 2003
http://nws.cs.ucsb.edu/ewiki/. 3.3.1

[Olivier et André 2003] FESTOR OLIVIER et SCHAFF ANDRÉ, *Standards pour la gestion des réseaux et des services (Traité IC2, série Réseaux et Télécoms)*, Lavoisier, 2003. 3.3.2

[OSGi Alliance 2002] OSGI ALLIANCE, *The OSGi Alliance web site*, 2002
http://www.osgi.org/. 2.2.3, 2.2.3

[Parrend et Frénot 2006b] PIERRE PARREND et STÉPHANE FRÉNOT, *Secure Component Deployment in the OSGi Release 4 Platform*, Technical Report n°0323, INRIA Rhône-Alpes, 2006. 3.6.1

[Parrend et Frénot 2007b] PIERRE PARREND et STÉPHANE FRÉNOT, « Supporting the Secure Deployment of OSGi Bundles », *First IEEE Workshop on Adaptive and DependAble Mission- and bUsiness-critical mobile Systems (ADAMUS) at WoWMoM*, Helsinki, Iceland, 18 june 2007. 3.6.1

[Parrend et Frénot 2008b] Pierre Parrend et Stéphane Frénot, « Component-based Access Control : Secure Software Composition through Static Analysis », *7th Symposium on Software Composition (SC'2008) at ETAPS*, 2008. 3.6.1

[pic 2007] *Pico Container*, 2003–2007
http://www.picocontainer.org/. 2.2.3

[Projet Amigo 2007] Projet Amigo, *Ambient intelligence for the networked home environment, IST-004182 Framework Programme 6*, 2004-2007
http://www.hitech-projects.com/euprojects/amigo/. 3.7

[Projet DARTS 2002b] Projet DARTS, *Déploiement et Administration de Ressources, Traitements et Services*, 2002
http://darts.insa-lyon.fr/index.html.en. 1.2, 3.2.3

[Projet LISE 2010] Projet LISE, *Liability Issues in Software Environment*, 2008–2010
http://www-anr-ci.cea.fr/scripts/home/publigen/content/templates/show.asp?P=145&L=FR&ITEMID=17/. 1.2, 3.6.2

[Projet MUSE 2007] Projet MUSE, *MultiServices EveryWhere, IST-026442 Framework Programme 6*, 2004–2007
http://www.ist-muse.org/. 1.2, 3.3

[Rellermeyer et al. 2007] Jan S. Rellermeyer, Gustavo Alonso et Timothy Roscoe, « R-OSGi : Distributed Applications through Software Modularization », *Proceedings of the ACM/IFIP/USENIX 8th International Middleware Conference (Middleware 2007)*, November 2007. 3.2.2

[Rellermeyer et Alonso 2007a] Jan S. Rellermeyer et Gustavo Alonso, « Concierge : a service platform for resource-constrained devices », *EuroSys '07 : Proceedings of the 2007 conference on EuroSys*, p. 245–258, ACM Press, New York, NY, USA, 2007. 3.6.1

[Royon et al. 2004] Yvan Royon, Stéphane Frénot et Antoine Fraboulet, *Approche orientée composant d'une pile réseau*, Technical Report n°0298, INRIA Rhône-Alpes, 07 2004. 2.3

[Royon et al. 2005] Yvan Royon, Stéphane Frénot et Antoine Fraboulet, « Mynus : une pile réseau dynamique », *Journées Composants*, Le Croisic, France, april 2005. 2.3

[Royon et al. 2006] Yvan Royon, Stéphane Frénot et Frederic Le Mouel, « Virtualization of Service Gateways in Multi-provider Environments », *Proceedings of Component Based Software Enginering conference (CBSE'2006)*, Vasteras, Stockholm, Sweden, juin 2006. 3.4.3

[Royon et Frénot 2007b] Yvan Royon et Stéphane Frénot, « Multiservice Home Gateways : Business Model, Execution Environment, Management Infrastructure », *IEEE Communications Magazine*, p. 122–128, vol. 45, October 2007, <a href="mailto :stephane.frenot@insa-lyon.fr ?subject=Requesting a copy of IEEE Comm Mag 2007">request a copy</A>. 3.4.3

[Royon 2007] Yvan Royon, *Environnements d'exécution pour passerelles domestiques*, Thèse de doctorat, INSA de Lyon, December 2007. 1.2, 3.4.3

[Schwartz et al. 2006] Randal L. Schwartz, Tom Christiansen, Stephen Potter et Larry Wall, *Programming Perl*, Oreilly & Associates Inc, 2006. 2.2.1

[SCK et Wyant 1994] Ann Wollrath Samuel C. Kendall Jim Waldo et Geoff Wyant, *A Note on Distributed Computing*, rapport technique n°SMLI TR-94-29, Sun Microsystems, November 1994. 2.1.2, 3.2.2

[Sklar et Trachtenberg 2006] David Sklar et Adam Trachtenberg, *PHP Cookbook*, O'Reilly Media, 2006. 2.2.1

[Smith et Nair 2005] Jim Smith et Ravi Nair, *Virtual Machines : Versatile Platforms for Systems and Processes*, Morgan Kaufmann / Elsevier, San Francisco, CA, June 2005. 3.4.2

[Specification 1998a] Sun Specification, *Java DataBases Connectivity*, 1998 `http://java.sun.com/javase/technologies/database/`. 2.2.2

[Specification 1998b] Sun Specification, *Java Messaging Service*, 1998 `http://java.sun.com/products/jms/`. 2.2.2

[Specification 1998c] Sun Specification, *Java Naming and Directory Interface*, 1998 `http://java.sun.com/products/jndi/docs.html#12`. 2.2.2

[Specification 1998d] Sun Specification, *Java Transaction API*, 1998 `http://java.sun.com/javaee/technologies/jta/index.jsp`. 2.2.2

[Stevens et Wright 1995] W. Richard Stevens et Gary R. Wright, *TCP/IP illustrated (vol. 2) : The implementation*, Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1995. 2.1

[Stroustrup 2003] Bjarne Stroustrup, *Le langage C++*, Pearson Education, 2003. 2.2.1

[Thomas et Fowler 2004] David Thomas et Chad Fowler, *Programming Ruby : The Pragmatic Programmer's Guide*, Pragmatic Bookshelf, 2004. 2.2.1

[VM List 2008] Wikipedia VM List, *List of Java virtual machines*, 2008 `http://en.wikipedia.org/wiki/List_of_Java_virtual_machines`. 3.5.2

[VMware 2006] VMware, *The VMware project*, since 2006 `http//www.vmware.com`. 3.4.2

[Welch 2008] Brent B. Welch, *Practical Programming in Tcl and Tk*, Prentice Hall PTR, 2008. 2.2.1