Année 2008

Curriculum vitæ

pour une demande de qualification

AUX POSTES DE PROFESSEUR DES UNIVERSITÉS

spécialité informatique

par

Stéphane FRENOT

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

Chapter 1

CV

1.1 Civilité

Ce CV décrit mes activités de recherche et d'enseignement depuis ma thèse en novembre 1998.

Stéphane Frénot
13 av. des Sports
F-69500 Bron
Né le 27 septembre 1970 à Strasbourg (67)
Marié 3 enfants
`http://perso.citi.insa-lyon.fr/sfrenot`
stephane.frenot@insa-lyon.fr
Bénéficiaire de la PEDR depuis 2004

1.1.1 Expériences professionnelles

Depuis septembre 1999

- Maître de conférences au département télécommunications, services et usages de l'INSA de Lyon
- Membre du laboratoire CITI, responsable du thème Middleware
- Membre du projet Ares / Amazone

1998-1999 ingénieur de recherche

- Département informatique de l'INSA Lyon

1997-1998 ingénieur de développement

- Société Al'X, responsable d'un projet de développement d'application de gestion de dossiers patients en milieu hospitalier

1.1.2 Diplômes

Novembre 1998 : Thèse en informatique Université Lyon I
"Nouvelles technologies pour la gestion et la représentation de l'information médicale"
Directeur de recherche : Pr. André Flory
Rapporteurs : Pr. Marie-France Bruandet, Pr. Guy Gouardère
Jury : Pr. Gérard Duru, Pr. Jean-Marc Geib, Pr. Maurice Laville, Dr. Alain Mercatello, Mr. Gérard Alix

Septembre 1993 : DEA informatique INSA Lyon
"Une nouvelle interface pour la gestion des prescriptions médicales basées sur des techniques d'apprentissages"
Laboratoire LISI, INSA Lyon, Encadré par Pr. André Flory

Juin 1993 : Diplôme d'ingénieur INSA Lyon
Département Informatique

1.2 Recherche

1.2.1 Publications

La table 1.1 présente une synthèse de mes publications. Les publications récentes portent sur les approches à composants et leur gestion.

4 journaux internationaux
14 publications en conférences internationales
3 journaux nationaux
16 publications nationales
6 workshops
4 présentations invitées
10 rapports de recherche et techniques INRIA

Tab. 1.1 – Synthèse des publications

1.2.1.1 Journaux internationaux

[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. 1.2.2.3, 2, 2.5

[Frénot et Laforest 1999] Stéphane Frénot et Frédérique Laforest, « Medical Records Management Systems : Critics and New Perspectives », *Methods of Information in Medicine*, p. 89–95, vol. 38, n°2, 1999. 1.2.2

[Parrend et Frénot 2009] Pierre Parrend et Stéphane Frénot, « Security Benchmarks of OSGi Platforms : Towards Hardened OSGi », *To be published in Software Practice and Experience*, 2009.

[Royon et Frénot 2007] Yvan Royon et Stéphane Frénot, « Multiservice Home Gateways : Business Model, Execution Environment, Management Infrastructure », *IEEE Communications Magazine*, p. 122–128, vol. 45, October 2007. 1.2.2.3, 2, 2.1

1.2.1.2 Publications en conférences internationales avec comités de lecture

[Al Masri et Frénot 2001] Nada Al Masri et Stéphane Frénot, « Speech Recognition Integration in Medical Information System », *MedInfo*, London, England, October 2001.

[Ben Hamida et al. 2008] A. Ben Hamida, F. Le Mouël, S. Frénot et M. Ben Ahmed, « A Graph-based Approach for Contextual Service Loading in Pervasive Environments », *Proceedings of the 10th International Symposium on Distributed Objects and Applications (DOA'2008), Lecture Notes in Computer Science*, vol. 5331, p. 589–606, Springer Verlag, Monterrey, Mexico, novembre 2008.

[Flory et Frénot 1999] André Flory et Stéphane Frénot, « An Intelligent System for Drug Prescription Support », *Intelligent Systems : ISCA 5th International Conference. International Society for Computers and Their Applications - ISCA*, Denver, CO, USA, 1999.

[Frénot et al. 1995] Stéphane Frénot, Frédérique Laforest et André Flory, « An Intelligent System to Help Expert Users : Application to Ddrug Prescription », *7th int conf on Artificial Intelligence and Expert Systems Applications (EXPERSYS'95)*, p. 61–66, San Francisco, CA, USA, November 1995. 1.2.2

[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. 1.2.2.3, 2, 2.2

[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. 1.2.2.3

[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.

[Ibrahim et al. 2007b] Noha Ibrahim, Frédéric Le Mouël et Stéphane Frénot, « Automatic Service-Integration Framework for Ubiquitous En-

vironments », *Proceedings of the International Conference on Mobile Ubiquitous Computing, Systems, Services and Technologies (UBICOMM'2007)*, Papeete, French Polynesia (Tahiti), France, November 2007.

[Laforest et al. 1996] Frédérique Laforest, Stéphane Frénot et André Flory, « A New Approach For Hypermedia Medical Records Management », *13th Int Congress Medical Informatics Europe (MIE'96)*, p. 1042–1046, Copenhagen, Danemark, 19-22 August 1996. 1.2.2

[Laforest et al. 1998] Frédérique Laforest, Stéphane Frénot et André Flory, « A Hypermedia-based Medical Records Management System », *MedInfo'98*, Seoul, Korea, August 1998. 1.2.2

[Parrend et al. 2007] Pierre Parrend, Samuel Galice, Stéphane Frénot et Stéphane Ubeda, « Identity-Based Cryptosystems for Enhanced Deployment of OSGi Bundles », *SECUREWARE'2007*, Valencia, Spain, October 2007. 1.2.2.3, 2, 2.3

[Parrend et Frénot 2006] Pierre Parrend et Stéphane Frénot, « A Security Analysis for Home Gateway Architectures », *International Conference on Cryptography, Coding and Information Security, CCIS 2006, November 24-26, Venice, Italy*, 2006. 1.2.2.3

[Parrend et Frénot 2008] Pierre Parrend et Stéphane Frénot, « Classification of Component Vulnerabilities in Java Service Oriented Programming (SOP) Platforms », *Proceedings of Component Based Software Enginering Conference (CBSE'2008)*, *Lecture Notes in Computer Science*, vol. 5282, Springer Verlag, Karlsruhe, Germany, octobre 2008.

[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. 1.2.2.3, 2, 2.4

[Royon et al. 2007] Yvan Royon, Pierre Parrend, Stéphane Frénot, Serafeim Papastefanos, Humberto Abdelnur et Dirk Van de Poel, « Multi-service, Multi-protocol Management for Residential Gateways », *BroadBand Europe*, Antwerp, Belgium, december 2007. 1.2.2.3

1.2.1.3 Journaux nationaux

[Ben Hamida et al. 2008] 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.

[Laforest et al. 2002] Frederique Laforest, Stéphane Frénot et Nada Al Masri, « Dossier médical semi-structuré pour des interfaces de saisie multi-modales », *Revue Documents numériques*, vol. 6, n°1-2, 2002, Numéro spécial "Les Dossiers numériques".

[Parrend et Frénot 2009] Pierre Parrend et Stéphane Frénot, « Composants, Services et Aspects : techniques et outils pour la vérification », *Revue l'Objet*, 2009.

1.2.1.4 Conférences nationales avec comités de lecture

[Al Masri et Frénot 2002] Nada Al Masri et Stéphane Frénot, « Instrumentation dynamique d'applications à base d'EJB », *Journées Composants*, Grenoble, France, October 2002. 1.2.2.3

[Ben Hamida et al. 2008] 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.

[Delestre et al. 2002] Nicolas Delestre, Stéphane Frénot, Stéphane Mottelet et Michel Vayssade, « Distributed PolyTeXML : Une nouvelle plate-forme de partage d'items didactiques », *TICE'2002*, Lyon, France, 2002.

[Frénot et Avin-Sotres 2001] Stéphane Frénot et Miguel Avin-Sotres, « Migration et Déploiement automatique de services », *Journées Composants (JC'2001)*, Besançon, France, 2001. 1.2.2.3

[Frénot et Laforest 1998] Stéphane Frénot et Frédérique Laforest, « Un système de gestion d'architextes », *Congrès INFORSID 1998*, Montpellier, France, mai 1998. 1.2.2

[Frénot et Laforest 2001] S. Frénot et F. Laforest, « ToutoPhone : fournir des services de haut niveau sur le téléphone », *MS3G'2001 Colloque sur 'Mobiles-services et réseaux mobiles de 3ème Génération*, Lyon, France, 2001.

[Frénot et Stefan 2004a] Stéphane Frénot et Dan Stefan, « Instrumentation de plates-formes de services ouvertes - Gestion JMX sur OSGi », *UbiMob*, Nice, France, jun 2004. 1.2.2.3

[Frénot et Stefan 2004b] Stéphane Frénot et Dan Stefan, « M-OSGi : Une plate-forme répartie de services », *Notere*, Saïdia, Maroc, jun 2004. 1.2.2.3

[Frénot 2004] Stéphane Frénot, « Gestion du déploiement de composants sur réseau P2P », *DECOR*, Grenoble, October 2004. 1.2.2.3

[Gorce et al. 2001] Jean-Marie Gorce, Stéphane Frénot, Virginie Crespo et Stéphane Ubéda, « Modélisation de la propagation en environnement INDOOR dans la bande de fréquence UHF », *ICISP*, Agadir, Maroc, 2001.

[Ibrahim et al. 2006] Noha Ibrahim, Frédéric Le Mouël et Stéphane Frénot, « Intégration négociée de services dans les systèmes distribués », *Journées Composants (JC'2006)*, Perpignan, France, October 2006.

[Le Mouël et al. 2005] Frédéric Le Mouël, Noha Ibrahim et Stéphane Frénot, « Interface Matching and Combining Techniques for Services

Integration », *3er Congresco Nacional de Ciencias de la Computacion*, FCC-BUAP, Puebla, Mexico, 2005.

[Royon et al. 2005] Yvan Royon, Stéphane Frénot et Antoine Fraboulet, « Mynus : une pile réseau dynamique », *Journées Composants*, Le Croisic, France, april 2005. 1.2.2.3

[Royon et Frénot 2006a] Yvan Royon et Stéphane Frénot, « Architecture de gestion de passerelles domestiques de services », *GRES*, Bordeaux, France, mai 2006. 1.2.2.3

[Royon et Frénot 2006b] Yvan Royon et Stéphane Frénot, « Un environnement multi-utilisateurs orienté service », *CFSE'2006*, Le Canet, France, October 2006. 1.2.2.3

1.2.1.5 Workshops

[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.

[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. 1.2.2.3

[Frénot 2003a] Stéphane Frénot, « JMX Management », *Fractal Workshop at ObjectWeb Consortium meeting*, Inria, Grenoble, France, 2003.

[Frénot 2003b] Stéphane Frénot, « System auto-configuration on top of OSGi », *Architecture Meeting at ObjectWeb Consortium*, lip6, Paris, juillet 2003. 1.2.2.3

[Ibrahim et al. 2005] Noha Ibrahim, Frédéric Le Mouël et Stéphane Frénot, « Automatic Negotiated Integration of Services in Pervasive Environments », *Middleware for Web Services Workshop (MWS 2005)*, Enschede, The Netherlands, 19 September 2005.

[Ibrahim et al. 2006] Noha Ibrahim, Frédéric Le Mouël et Stéphane Frénot, « Techniques d'intégration de services dans les environnements distribués », *Actes du 7 ème atelier sur les Objets, Composants et Modèles dans l'ingénierie des Systèmes d'Information (OCM-SI'2006) at INFORSID*, Hammamet, Tunisie, May 2006.

[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.

[Parrend et Frénot 2007] 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. 1.2.2.3

[Parrend et Frénot 2008] 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.

1.2.1.6 Rapports techniques

[Al Masri et Frénot 2002] Nada Al Masri et Stéphane Frénot, *Dynamic instrumentation for the management of EJB-based applications*, Technical Report n°4481, INRIA Rhône-Alpes, 2002. 1.2.2.3

[Fleury et Frénot 2003] Eric Fleury et Stéphane Frénot, *Building a JMX management interface inside OSGi*, Technical Report n°5025, INRIA Rhône-Alpes, 2003. 1.2.2.1, 1.2.2.3

[Frénot et al. 2002] Stéphane Frénot, Miguel Avin-Sotres et Nada Al-Masri, *EJB Components Migration Service and Automatic Deployment*, Technical Report n°4480, INRIA Rhône-Alpes, 2002. 1.2.2.3

[Parrend et Frénot 2006] Pierre Parrend et Stéphane Frénot, *Secure Component Deployment in the OSGi Release 4 Platform*, Technical Report n°0323, INRIA Rhône-Alpes, 2006. 1.2.2.3

[Parrend et Frénot 2007] Pierre Parrend et Stéphane Frénot, *Java Components Vulnerabilities - An Experimental Classification Targeted at the OSGi Platform*, Technical Report n°6231, INRIA Rhône-Alpes, 2007. 1.2.2.3

[Parrend et Frénot 2008] Pierre Parrend et Stéphane Frénot, *More Vulnerabilities in the Java/OSGi Platform : A Focus on Bundle Interactions*, Technical Report n°6649, INRIA Rhône-Alpes, 2008.

[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. 1.2.2.3

[Royon et Frénot 2007] Yvan Royon et Stéphane Frénot, *A Survey of Unix Init Schemes*, Technical Report n°0338, INRIA Rhône-Alpes, June 2007, https ://hal.inria.fr/inria-00155663.

1.2.1.7 Autres publications

[Al Masri et Frénot 2002] Nada Al Masri et Stéphane Frénot, « Unified and Automatic Enterprise JavaBeans Instrumentation », *Poster at DOA*, Irvine, CA, USA, Dec 2002. 1.2.2.3

[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.

[Frénot et al. 2003] Stéphane Frénot, Anis Krichen et Stéphane Ubéda, « Light Support for Dynamic and Pervasive Services on P2P Networks », *Ercim news No. 54, juille*, 2003.

[Frénot et Laforest 1998] Stéphane Frénot et Frédérique Laforest, « Modélisation du dossier médical multimédia distribué », *Carrefours de la fondation Rhône-Alpes futur*, Lyon, France, septembre 1998, (2nd prix de la fondation). 1.2.2

[Frénot et Papastefanos 2008] Stéphane Frénot et Serafeim Papastefanos, *Virtual Machines for Embedded Environments*, MUSE IST-6thFP-026442, public deliverable, DB3.6, january 2008.

[Frénot 2005] Stéphane Frénot, *Open Management Using OSGi Technology Enabled Services*, OSGi world Congress, developper session, October 2005.

[Ibrahim et al. 2009] N. Ibrahim, F. Le Mouël et S. Frénot, *Middleware Technologies for Ubiquitous Computing*, *Handbook of Research on Next Generation Networks and Ubiquitous Computing*, p. –, Handbook of Research on Next Generation Networks and Ubiquitous Computing, IGI Global Publication, 2009, to appear.

[Le Mouël et Frénot 2008] F. Le Mouël et S. Frénot (éd.), *Proceedings of the 3rd ACM International Workshop on Services Integration in Pervasive Environments (SIPE'2008)*, Sorrento, Italy, ACM Press, juillet 2008.

[Le Mouël et Frénot 2006] Frédéric Le Mouël et Stéphane Frénot (éd.), *1st IEEE International Workshop on Services Integration in Pervasive Environments at ICPS*, Lyon, France, IEEE Press, June 2006, `http://ares.insa-lyon.fr/sipe06/`.

[Le Mouël et Frénot 2007] Frédéric Le Mouël et Stéphane Frénot (éd.), *2nd IEEE International Workshop on Services Integration in Pervasive Environments (SIPE'2007) at ICPS*, Istambul, Turkey, IEEE Press, July 2007.

[Parrend et Frénot 2006] Pierre Parrend et Stéphane Frénot, « Dependability for Component Systems Deployment », *Poster at EuroSys Conference*, Leuven, Belgium, 18-21 April 2006. 1.2.2.3

[Projet DARTS 2002] Projet DARTS, *Déploiement et Administration de Ressources, Traitements et Services*, `http://darts.insa-lyon.fr/index.html.en`, 2002. 1.2.2

1.2.2 Activités de recherche

Pendant ma thèse, j'ai travaillé sur les systèmes d'information médicaux hospitaliers. Au début de mes travaux, les interfaces utilisateurs de systèmes large échelle[1] étaient principalement des interfaces X11 ou Windows. Au moment des balbutiements du Web, j'ai proposé d'avoir une approche orientée

[1] au sens grand hôpital ou centres hospitaliers comme les hospices civils de Lyon

documents pour la gestion de l'information médicale. Cette approche documentaire repose sur une architecture Web anticipant de deux années les architectures actuelles. J'ai défini le principe d'*architexte* pour la gestion des informations médicales. Un *architexte* est une collection d'extraits documentaires qui représentent l'intégralité du dossier médical s'ils sont regroupés. J'ai également développé l'architecture de gestion des *architextes* pour l'entreprise Al'X durant ma CIFRE. Cette activité s'est arrêtée lorsque BULL SA, qui commercialisait le produit, a revendu sa branche santé à une entreprise spécialisée dans le domaine médical ESCARE SA.

Ces travaux ont été présentés dans diverses publications : dans un journal international [Frénot et Laforest 1999], en conférences internationales [Laforest et al. 1996, Laforest et al. 1998, Frénot et al. 1995], en conférences nationales [Frénot et Laforest 1998] ainsi que dans des présentations locales comme [Frénot et Laforest 1998].

Pendant cette période, 1994-1997, j'ai développé en C++, Java et Perl ainsi que dans de nombreuses architectures orientées Web. Notre approche à base de serveurs d'objets C++ était précurseur des architectures J2EE à base de Java qui allaient sortir un ou deux ans après. De notre point de vue, l'architecture d'architexte correspond parfaitement à la tendance du Web2.0 qui est une reconstitution au niveau du client de consultation d'un ensemble de données issues de plusieurs sources. Le client Web se comporte alors comme un point de convergence local de l'information désirée par l'utilisateur.

Après ma thèse, j'ai participé à la création du laboratoire CITI, au montage du département Télécommunications Services et Usages[2] ainsi qu'à deux équipes de recherche INRIA, ARES et AMAZONES.

J'ai changé d'activité à ce moment en laissant de côté l'activité médicale et en me recentrant sur le domaine fonctionnel des composants. Les approches à composants visées étant focalisés sur la conception d'intergiciels pour systèmes dynamique et mobiles.

Les applications courantes nécessitent de nouvelles infrastructures pour gérer la distribution, la dynamique, l'intégration autonome de nouveaux services et de nombreuses activités de haut niveau qui ne sont pas ou peu prises en charge par les couches "classiques" des systèmes d'exploitation. Mon défi est de concevoir et d'implanter de nouveaux intergiciels, ou systèmes d'exploitation globaux intégrant ces nouveaux besoins.

J'ai initialement travaillé dans le cadre de l'architecture J2EE afin de proposer une méthode d'administration d'une fédération de serveurs d'applications. Les différents nœuds partagent l'exécution des EJB et nous pouvons faire migrer des EJB entre nœuds durant l'exécution. Le système que nous avons conçu intègre une description des ressources liées aux EJB afin de planifier leur déploiement et leur exécution. Ce travail a été partiellement réalisé dans le cadre du projet DARTS, Déploiement et Administration de Ressources, Traitements et Services. Le projet DARTS [Projet DARTS 2002] est une ACI de 2 ans.

[2] à partir de la seconde promotion entrante

DARTS est focalisé sur les approches pour la grille de calcul, qui est une thématique un peu éloignée des activités du laboratoire CITI. Le problème majeur est la taille croissante des architectures J2EE qui devenaient de plus en plus complexes à intégrer dans de petits environnement sans fil. Je me suis réorienté sur des environnements plus embarqués, petits et ambiants. J'ai étudié les propositions industrielles et académiques et me suis rapidement recentré sur la spécification OSGi. J'ai alors organisé le travail de l'équipe autour de cette spécification. Mon implication autour du framework OSGi porte sur 3 axes :

- Implantation de certaines spécifications
- Dissémination et enseignements autour de cette technologie
- Conception d'extensions non fonctionnelles à la plate-forme

1.2.2.1 Implantations intégrées à la communauté

La spécification OSGi est implantée par plusieurs solutions *open-source*, FELIX, EQUINOX ET KNOPFLERISH. Cependant aucune d'entre elles n'implante l'intégralité. J'ai fourni à la communauté FELIX une extension pour la supervision de passerelles OSGi ainsi qu'une implantation de la partie découverte automatique de services [Fleury et Frénot 2003].

1.2.2.2 Dissémination et enseignements

Depuis que j'utilise cette technologie, je suis convaincu des avantages substantiels qu'elle apporte dans le domaine du génie logiciel. J'ai monté en conséquent des cours à ce sujet. J'ai présenté le framework aux équipes de développement de MOTOROLA TOULOUSE en 2002, ainsi que dans diverses écoles et tutoriaux pour chercheurs (Ecotel'2002, Notere'2004, ING'2005, OSGi'2007, BBE'2007, ICAR'2008). J'interviens également dans des cours d'étudiants de cycle d'ingénieur et de Master recherche ; Département Télécommunication de l'INSA de Lyon dans une option de 4ème année, à l'ENSEIRB de Bordeaux ainsi qu'à CPE de Lyon.

1.2.2.3 Conception d'extensions non fonctionnelles à OSGi

En dernier point, mon activité la plus importante a été de proposer diverses extensions au framework OSGi. J'ai conçu et développé ces extensions dans les domaines suivants :

- Administration : comment gérer au mieux les passerelles (MOSGi et VOSGi)
- Sécurité : comment garantir la sécurité des composants (SOSGi)
- Embarquement : comment embarqué OSGi dans des environnements réduits (ROCS)
- Intégration du contexte : comment réagir aux modifications de l'environnement (POSGi, ANIs)

Le tableau 1.2 représente une synthèse des projets, des personnes qui y ont participé et de leur rôle. Il présente également les publications et et les financements associées à chaque projet.

Project, publications	**Theme**	**Participant**	**Fundings**
DARTS	Développement principal	Nada Al Masri (thèse)	DARTS ACI (99-2002)
[Brebner et al. 2005], [Frénot et Stefan 2004b], [Al Masri et Frénot 2002]	Migration d'EJB	Miguel Avin Sotres (Mastère recherche)	
[Frénot et Stefan 2004a], [Frénot et Balan 2003], [Frénot et al. 2002]	Evaluation de RMI	Tudor Balan (Mastère recherche)	
[Al Masri et Frénot 2002], [Al Masri et Frénot 2002], [Fleury et Frénot 2003], [Frénot et Avin-Sotres 2001]	OSGi Distribué	Dan Stefan (Ingénieur associé)	
POSGi	Distribution pair-à-pair de bundles	Dan Stefan (Ingénieur associé)	
[Frénot et Royon 2005], [Frénot 2004], [Frénot 2003b]		Tarek Tuurki (Mastère recherche)	
VOSGi	OSGi virtuel	Yvan Royon (thèse)	MUSE IP-FP6 (03-05)
[Royon et al. 2006], [Royon et Frénot 2006b], [Royon et Frénot 2006a], [Royon et al. 2005], [Royon et al. 2004]		Denis Beras (Ingénieur associé)	
SOSGi	Secured OSGi	Pierre Parrend (thèse)	MUSE II IP-FP6 (05-07)
[Parrend et Frénot 2007], [Parrend et Frénot 2006], [Parrend et Frénot 2006], [Parrend et Frénot 2007], [Parrend et Frénot 2006], [Parrend et al. 2007]		Mathieu Chantrel (Ingénieur associé)	
MOSGi	Administration d'OSGi	Yvan Royon (thèse)	MUSE IP-FP6 (03-05)
[Royon et Frénot 2007], [Frénot et al. 2008], [Royon et al. 2007]		Denis Beras (Ingénieur associé)	

TAB. 1.2 – Synthèse des projets de recherche

Suite à cette montée d'expertise sur la plate-forme OSGi, j'ai été contacté par les personnes du consortium MUSE afin d'intégrer ce projet européen. L'objectif du projet était de standardiser les architecture de livraison de services ADSL au niveau de l'Europe. L'approche OSGi était un candidat potentiel dans le cadre de l'hébergement des services sur les boitiers aux domiciles des utilisateurs. Les projets MUSE et MUSE II nous ont permis d'élaborer de nouvelles extensions et de nouvelles approches autour des passerelles de services. Ces approches et extensions sont largement décrites dans mon document d'Habilitation. Je poursuis actuellement les travaux autour d'OSGi dans le cadre de l'ANR LISE (Liability Issues in Software Environment) dont l'objectif et de permettre le rapprochement entre les textes issus du droit du logiciel et les traces d'exécutions obtenues lors des pannes réelles des systèmes. Au cours de ces 3 projets, DARTS, MUSE, LISE, j'ai encadré un certain nombre d'étudiants dont la présentation est faite dans la section suivante.

1.2.3 Encadrements

Je suis responsable de l'équipe Middleware du CITI. Elle est actuellement composée de 2 maîtres de conférences, 5 étudiants en thèse, 2 ingénieurs et de quelques étudiants en stages. J'organise une fois par mois des réunions de travail où chaque personne présente son activité.

1.2.3.1 Co-Encadrement de thèse

Je co-encadre des thèses (cf. tableau 1.3) depuis que j'ai été recruté et je suis co-auteurs de l'ensemble des publications des étudiants.

Soutenance	**Etudiant**	**Situation actuelle**
Mar 05	Nada Al Masri "Modèle d'Administration des Systèmes Distribués à Base de Composants"	Professeur Associée, University of Waterloo, Canada
Déc 07	Yvan Royon "Environnements d'exécution pour passerelles domestiques"	Qualifié 27, Ingénieur R&D Alcatel Belgique
Déc 08	Pierre Parrend "Sécurité des approches à composants"	Post doc Karlsruhe

TAB. 1.3 – Encadrement de thèses

1.2.3.2 Encadrement de Masters recherche

Les masters encadrés ont creusé certaines idées des projets. Yvan Royon et Nada Al Masri ont continué en thèse. Le tableau 1.4 indique la liste des 8 masters en informatique que j'ai encadrés.

Année	**Etudiant**	**Mots clés**
2005	Tarek Tuurki	P2P, OSGi
2004	Radu Popa	Grid, Globus, OSGi
2004	Yvan Royon	Pile IP à composants
2003	Tudor Balan	Perfomance de RMI
2003	Anis Krichen	OSGi, Deployment
2001	Christophe DeVaux-Bidon	QoS, EJB
2000	Miguel Avin Sotres	Nommage des EJB
1999	Nada Al Masri	Supervision des EJB

TAB. 1.4 – Masters recherche

1.2.3.3 Ingénieurs Associés

Sur les supports budgétaires de DARTS, MUSE et MUSE II j'ai embauché 3 ingénieurs associés (c.f. tableau 1.5). En plus du travail de développement nécessaire, ils ont fourni un travail de documentation et de supervision de la qualité du code développé.

Année	**Nom**	**Projet, Mots-clés**
2007-2008	Mathieu Chantrel	Fond INRIA, UPnP
2005-2008	Denis Beras	MUSE, Supérvision, Sécurité
2003-2004	Dan Stefan	DARTS, OSGi, Distribution

TAB. 1.5 – Ingénieurs associés

1.2.3.4 Autres encadrements

J'ai encadré de nombreux étudiants en stage et en projet de fin d'études. Le tableau 1.6 en donne la liste.

Année	Etudiant	Origine	Mots-clés
2008	Stéphane Chevali	CPE	Interposition de code de trace
2007	Martin Schlienger, Xiaotian Li	INSA Telecom	ClassLoader Pair-à-pair
2006	Yoann Duclaux, Cyril Ganier	INSA Telecom	ClassLoader Pair-à-pair
2006	Benoit Prat, Arnaud Dumont	CPE	Etude des dépendances de classes
2005	Hayder Bassem	INSAT Tunis	Outil de gestion de dépendances de classes
2005	Nemeth Annamaria Feroiu	Lyon University	OSGi pair-à-pair
2005	Baptiste Decroix	CPE	Machine virtuelle enfouie
2005	Baptiste Decroix, Mathias Bertschy	CPE	Avalon, Nano|Pico Container
2005	Aurelien Laurendon	INSA Telecom	TCP/IP à composant
2004	Oscar Bigu	University of Catalunya	UPnP, OSGi
2004	Gilles Erwan	INSA Telecom	Java embarqué
2004	Khaled Yousfi, Riadh ben Hariz	CPE	Supervision JMX
2004	Djibril Kone	IMAG Grenoble	Instrumentation JMX

TAB. 1.6 – Autres encadrements

1.2.4 Dissémination

1.2.4.1 Edition de proceedings

Je suis co-éditeur de trois proceedings du workshop SIPE.
- F. Le Mouël and S. Frénot, Proceedings of the 3rd IEEE International Workshop on Services Integration in Pervasive Environments (SIPE'2008)
- F. Le Mouël and S. Frénot, Proceedings of the 2nd IEEE International Workshop on Services Integration in Pervasive Environments (SIPE'2007)
- F. Le Mouël and S. Frénot, Proceedings of the 1st IEEE International Workshop on Services Integration in Pervasive Environments (SIPE'2006)

1.2.4.2 Comité de programme

Je suis membre du comité de programme de 3 conférences internationales, d'une conférence nationale et de 3 workshops internationaux. Je suis également reviewer dans 2 journaux internationaux Oxford Journal et le journal koréen ETRI.

- **Journaux**
  - Oxford Computer Journal
  - ETRI Journal
- **Conférences Internationales**
  - 11th IFIP/IEEE Network Operations and Management Symposium (NOMS 2008)
  - 10th IFIP/IEEE International Symposium on Integrated Network Management (IM 2007)

  - 6th International Conference on Software Engineering Research, Management and Applications (SERA 2008)
- **Workshops Internationaux**
  - 19th IFIP/IEEE International Workshop on Distributed Systems : Operations and Management 2008 (DSOM 2008)
  - 2nd International workshop on Adaptive and DependAble Mobile Ubiquitous Systems (ADAMUS 2008 - WoWMoM 2008)
  - 1st IEEE WoWMoM Workshop on Adaptive and DependAble Mission- and bUsiness-critical mobile Systems (ADAMUS 2007 - WoWMoM 2007)
- **Conférence nationale**
  - Nouvelles technologies de la répartition (Notere) - 2005, 2006, 2007, 2008

1.2.5 Financements

J'ai participé à un projet européen MUSE, qui a été reconduit en MUSE II, sur une durée de 4 ans. Sur la phase II j'étais le représentant INRIA sur le projet. J'ai piloté l'ACI DARTS, et je participe aux projets LISE. Le tableau 1.7 résume ces informations. La suite de cette section explique ce qui a été fait dans chaque projet.

Years	Acronym	Role	Funds provider
01-03	DARTS	**Leader** for the project (2 lab teams)	ACI French Ministry of Research
03-05	MUSE I	Participant	European project
05-07	MUSE II	**INRIA representative** and participant	European project
08-10	LISE	Participant	ANR French Ministry of Research

TAB. 1.7 – Contrats

2001-2003 DART ACI GRID[3] **Titre :** Déploiement et Administration de Ressources Traitements et Services
Responsabilité : Responsable du projet, lequel implique deux équipes dans deux laboratoires
Partenaires : LISI (LIRIS)
Description : DARTS a pour objectif de définir une architecture de déploiement et d'administration de services pour la grille. Les calculs sont distribués sur un réseau hétérogène de périphériques. Chaque périphérique déclare ses disponibilités et ses ressources. Un ordonnanceur déploie les calculs sur les nœuds disponibles. Nous utilisons la technologie OSGi pour administrer les différents nœuds. Chaque calcul est emballé dans un *bundle* OSGi déclarant le service

[3] `http://darts.insa-lyon.fr/index.html.fr`

correspondant. L'autre équipe de recherche s'est plus focalisé sur la gestion de la donnée dans ce contexte. Ils ont proposé une mise en œuvre du *toolkit* Globus v3 au dessus d'OSGi. Le projet a démontré les bonnes potentialités d'OSGi pour le contexte d'utilisation dans la grille.

2003-2007 MUSE I et II, projets européens du FP6 de type IP[4]
Titre : MUlti-AcceSs Everywhere
Responsabilité : Participant au projet et Représentant de l'INRIA en phase II. Dans le workpackage spécifique, nous avons collaboré avec THOMSON, ST MICROELECTRONICS et NTUA [5] en Grèce.
Partenaires : MUSE est un projet intégré qui regroupe une soixantaine de partenaires[6].
Description : Le projet a pour objectif la conception des réseaux de livraison de services ADSL pour 2010. Il couvre toute la chaîne économique du fournisseur de services à la passerelle domestique. Dans notre *workpackage*, nous sommes focalisés sur la spécification d'une passerelle de service au niveau matériel d'une part et au niveau système d'exploitation d'autre part. Notre expertise d'OSGi nous a permis de faire des test d'intégration dans des architectures industrielles telles que proposées par `Thomson`. Nous avons proposé dans le projet les extensions présentées dans ce mémoire afin de participer à l'élaboration du modèle économique multi-services, multi-fournisseurs préconisé.
Fonds : MUSE nous a permis de financer une thèse de doctorat sur 3 ans et 1 ingénieur de recherche sur les 2 phases du projet. Le montant attribué à l'équipe était de 110k euros par phases du projet.

2008-2010 LISE, projet ANR[7] **Titre :** Liability Issues In Software Engineering
Responsabilités : Participant
Partenaires : INRIA (Lyon, Grenoble), LIG (Grenoble), SUPELEC (Paris), DANTE (University of Versailles), PRINT (University of Caen)
Description : Le projet cherche à garantir l'infalsification des traces générées par des passerelles de service sécurisées. Quand survient une panne, la cause doit pouvoir être identifiée de manière univoque.

1.3 Responsabilités administratives

Lors de mon recrutement en 1999 j'ai activement participé au montage à la fois du département d'enseignement et du laboratoire de recherche. Mes responsabilités administratives ont été et sont encore assez importantes dans les trois facettes de notre métiers : recherche, enseignement et gestion administrative et technique.

[4] `http://www.ist-muse.org/`
[5] National Technical University of Athens
[6] `http://www.ist-muse.org/partners.html`
[7] `http://tinyurl.com/4l8j57`

Parmis celles-ci, j'étais membre élu de la commission de spécialiste INSA 27 de 2004 à 2008.

J'ai défini et monté l'intégralité du système d'information du département d'enseignement : les salles machines, la salle serveur, les intranets et les extranets. J'ai pris en charge ces tâches en attendant que le département ait le personnel requis.

La liste suivante résume mes activités administratives.

- **Au niveau INSA**
  - Membre élu de la commission de spécialistes 27
- **Au laboratoire CITI**
  - Co-fondateur du laboratoire
  - Responsable de l'équipe middleware du projet INRIA Ares (2003-2007)
  - Responsable de l'équipe middleware du projet INRIA Amazones (2008-)
- **Au département Télécommunications services et usages**
  - Construction des cours à partir de la 2nde promotion entrante
  - Membre du conseil de département de 1998 à 2001
  - Responsable et concepteur des projets Innovant de la formation
- **Environnement technique**
  - Responsable pendant 3 ans du système d'information du département
  - Administration et développement des applications de l'intranet du département : emploi du temps, gestion des contacts industriels, des outils de réservation, des outils d'information internes.

1.4 Enseignements

Durant ma thèse, ma convention CIFRE ne m'a pas permis d'avoir un gros volume d'enseignement initial. Mais j'enseignais dès 1995 des technologies en avance de phase comme le Web, Java et les intergiciels.

En tant que maître de conférences d'un nouveau département d'enseignement j'ai monté les cours suivants : TCP/IP, Middleware, Sécurité, Programmation Java, Génie logiciel. J'ai monté les cours magistraux, les travaux dirigés et les séances de travaux pratiques. Je réalise également quelques interventions autour d'OSGi, des systèmes pairs-à-pairs, des approches à composants et de la programmation agile. Mon activité liée aux enseignement peut se se diviser en trois domaines. Les enseignements classiques, les enseignements ponctuels, les montages de projets et d'options.

Tous mes cours sont depuis mon recrutement accessibles sur ma page Web. `http://perso.citi.insa-lyon.fr/sfrenot/courses.html`. Je détaille dans la suite les différentes catégories de cours en joignant des exemples de présentations associées.

1.4.1 Enseignements conventionnels

Au cœur de mes enseignements conventionnels, je regroupe les enseignements à base de cours/td/tp sur les matières enseignées au départements Télécommu-

nication. Le tableau suivant présente une synthèse de ces enseignements.

Matière	Année	Niveau	Cours/TD/TP
TCP/IP	2000	4ème année ingénieur	8h/8h/32h
Internet/Intranet	2000-2002	4ème année ingénieur	8h/4h/8h
Introduction à Java	2000-2003	3ème année ingénieur	8h/4h/8h
Programmation Objet Avancée	2001-2003	4ème année ingénieur	8h/4h/8h
Systèmes distribués et intergiciels	2000-Aujourd'hui	4ème année ingénieur 5ème année école CPE	28h/12h/24h
Java	2004-Aujourd'hui	3ème année ingénieur	2h/26h/20h
Plates-formes de services ouvertes	2005-2006	Mastère recherche	6h
Fondamentaux des systèmes distribués	2007-2008	Mastère recherche	6h

TAB. 1.8 – Liste des cours conventionnels

J'ai été responsable de tous ces enseignements, avec un encadrement de un à deux ATER ou vacataire pour la gestion des TP ou des interventions en TD. Le cours de Java était initialement présenté en deux sessions en 3 et 4ème années (initiation et programmation avancée). J'ai réalisé, avec les trois enseignants intervenant dans la matière, une refonte sous la forme de TD machine. L'intégralité des cours sont disponibles sur le site de gestion de cours de l'INSA de Lyon `http://moodle.insa-lyon.fr`. J'ai inclus en annexe les transparents du cours d'introduction à Java.

1.4.2 Enseignements ponctuels

Les enseignements ponctuels sont des enseignements que j'ai monté de manière indépendante des attentes de la formation principale. Ils s'agit d'intervention ponctuelles associés soit à mon domaine de recherche (par exemple OSGi), soit à des problématiques qui me semblaient relever d'un intérêt quelconque. Il s'agit la plupart du temps de cours de 3h ou 4h que je présente soit dans le cadre de séminaires soit dans des options. Le tableau suivant résume les quelques présentations que j'ai faites.

Titre	Public			
OSGi	Motorola Toulouse, professionnel		2003	2jours
	ENSEIRB Bordeaux		2005-2008	1jour
	École d'été ICAR		2008	3h
	École d'hivers BroadBand Europe (Anglais)		2007	3h
	Ecole d'été OSGi user group, supervision		2007	2h
	Option Systèmes pervasifs, 4ème ingénieur		2005-2008	1jour
Histoire des systèmes d'exploitation	3ème année ingénieur	2003-2008	2h de cours	
Linux Party	3ème année ingénieur	2002-2008	2x2h de TD	
eXtreme programming	4ème année ingénieur	2000-2008	2h de cours	
Réseaux Pairs-à-pairs	4ème année ingénieur	2003-2008	4h	
	Mastère recherche	2004-2007	4h	
Supervision JMX	Séminaire Fractal	2003	2h	

TAB. 1.9 – Liste des cours ponctuels

Parmi l'ensemble de ces présentations, j'insère en annexe les transparents du cours sur l'eXtreme programming. Ce cours ne correspond ni à mes activités de recherche, ni à mes assignations de formation initiale (intergiciel et programmation Java). C'est une intervention que j'ai monté suite à la lecture d'un livre sur cette approche pour la gestion du développement logiciel. J'ai monté 2h de cours que j'ai initialement placé comme dernier cours sur les intergiciels. Depuis, cette intervention se fait dans le cadre des projets innovants, présentés plus loin dans ce document.

1.4.3 Montage de projets et options

En arrivant au département Télécommunications de l'INSA de lyon, j'ai pris en charge les cours de base de la 3ème et de la 4ème année (Systèmes distribués et Java). J'ai préparé pour l'année suivante une option sur la sécurité des sytèmes d'informations. J'ai contacté cinq intervenants extérieurs et j'ai orchestré une demi-option de 30 heures d'enseignements en 2001. Cette option a ensuite été prise en charge par un collègue du département d'enseignement.

En 2004, j'ai proposé le montage d'une activité de projet pour les étudiants autour de l'innovation. J'ai monté un dossier pour la mise en place de projets dits Innovants. Les projets innovants sont des projets montés pas les étudiants autour d'idées qu'ils amènent eux-mêmes. Les étudiants gèrent ces projets au cours du second semestre de la 4ème année pour un volume de 250heures équivalent TD par équipe. Les structures d'équipes sont entièrement libre de 3 à 10 étudiants. Les projets se déroulent en 2 phases. Une première phase (février-mars) correspond au dépôt d'un cahier de type OSEO/anvar décrivant leur projet. Ce cahier est rapporté par des personnes extérieures (collègues de recherche,

contacts industriels). Suite à cette première évaluation, le projet le moins bien noté est arrêté et les étudiants de l'équipe sont répartis dans les projets encore en course. Les projets sont maquettés lors d'une seconde phase (mars-juin). A l'issue de ce maquettage/prototypage, les projets sont présentés lors de soutenances auprès des enseignants du département, les trois meilleurs projets sont présentés lors d'un forum entreprise qui a lieu en fin d'année.

J'ai défini et j'orchestre l'intégralité du mode de fonctionnement de ces projets. En effet, les équipes de projet sont encadrées par deux intervenants : un tuteur technique et un tuteur de gestion de projet. Tous les projets sont encadrés par des règles administratives de gestion, similaire aux projets ANR, c'est à dire que les étudiants doivent déposer régulièrement des documents sur des sites web, maintenir des sites de présentation et réaliser des rapports de progression.

L'objectif de ces projets est avant tout de former les étudiants à une approche de la gestion d'équipe et de la gestion de projets (organisation de tâches, structuration de la progression), avec un fort pouvoir moteur sur les étudiants car ils sont les propres acteurs de leurs innovations. L'organisation, l'historique et les cours associés aux projets innovations sont consultables sur ce site web que je maintiens entièrement `http://tc-net2.insa-lyon.fr/lutece/jsp/site/Portal.jsp?page_id=36`. Je joins ici un document de présentation des Projets Innovants qui a circulé au sein de l'INSA pour montrer le principe à des partenaires potentiels. Dans ce contexte, j'ai fait intervenir des personnes d'IDEAS labs à grenoble ainsi que la cellule R&D d'Atos Worldline qui présentent leur activité de recherche et développement à nos étudiants. Enfin je fait intervenir deux personnes de l'INPI dans le cadre de la brevètabilité et de la protection du logiciel. La description détaillé du fonctionnement des Projets Innovants se trouve en annexe.

1.4.4 Résumé sur mes activités d'enseignements

J'enseigne depuis 10 ans, et comme le montre cette dernière section j'ai monté un grand volume de cours. Les structures de ces cours sont ma contribution. Jusqu'à l'optention de la prime d'encadrement doctorale, j'ai eu des volumes d'enseignement de l'ordre de 280 heures. Le département étant en création j'ai eu une assez grande liberté pour adapter mes cours pendant ces dix ans. J'ai également eu la grand chance de pouvoir être une force de proposition en donnant les volumes et les modalités des cours que je donnais. Cette souplesse et la confiance accordée par les trois directions successives du département m'ont permis de former des étudiants de qualité pendant ces années. Enfin les projets innovants sont une vraie image de savoir faire spécifique au département qui est souvent présenté aux personnes exterieures : commission des titres d'ingénieur, présentation du département au étudiants en seconde année du tronc commun INSA, présentation aux entreprises... Enfin je souligne que je suis également intervenu sur des cours pour des formations professionnelles (formation Unix, Java), ainsi que dans le cadre de séminaires en anglais comme pour l'école d'hiver Broadband Europe.

1.4.5 Exemples de supports de cours

- Java
- eXtreme Programming
- Documentation de présentation des Projets Innovants

Présentation de Java

Stéphane Frénot – Frédérique Laforest – Frédéric Le-Mouël
2006
INSA lyon
Département Télécommunication Services & Usages

Langage de programmation

Write once, run anywhere

Java ?

- Printemps 90 : Naughton, Gosling et Sheridan :
  - *"Le consommateur est le centre du projet, il faut construire un environnement de petite taille avec une petite équipe et intégrer cet environnement dans une nouvelle génération de machines : des ordinateurs simples pour des gens normaux."*
- Printemps 91 : Microprocesseur grand public.
  - La "Green Team" prototype une machine de pilotage de l'électroménager
- Août 91 : Gosling développe Oak
- Août 92 : ----------duke----------> 

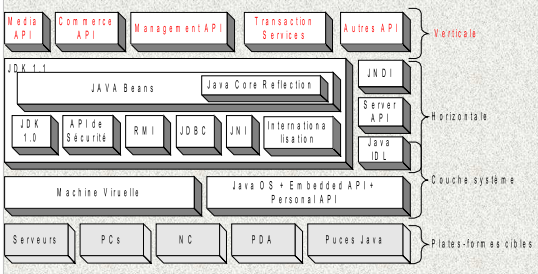
Java ?

- Eté 1993 : Sté. "First Person" est en train de couler
  - Eric Schmidt (Sun) demande une adaptation au Net
    - Gosling : travaille sur le code
    - Naughton : cherche une application stratégique
- Janvier 1995
  - Oak ==> Java, HotJava
- Août 1995 - Première licence sur Netscape
- Janvier 1996 - JDK 1.0.1
- Fév. 97 - JDK 1.1
- Jan 99 - JDK 2.0 (aka 1.2)
- 2003 J2se 1.4, J2ee, J2me (Standard, Entreprise, Micro)

Ce que c'est !

- Une architecture technique
- Un langage OO
- Une bibliothèque de 5000 éléments

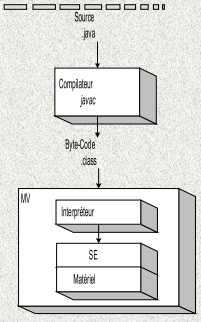
Plateforme Java : architecture



Java Client/Serveur : Nicolas, Avare, Najman - Eyrolles

Machine Virtuelle : Emulateur

- Byte-Code/P-Code/J-Code
  - opcode : 1 octet pour l'instruction
  - 0,n opérandes
- MicroProcesseur logiciel
  - Jeu d'instructions
  - Registres (pc, optop, frame, vars)
  - Pile, Heap
  - Ramasse-miettes
  - Espace de stockage des méthodes
  - Tas de constantes
- ==> Compilé ou Interprété



Machine Virtuelle

- Système d'exploitation
- Isolation
- Autonomie

====> Avantages ?

Java est portable

- Le compilateur Java génère du *byte code.*
- La *Java Virtual Machine* (JVM) est présente sur Unix, Win32, Mac, OS/2, Netscape, IE, ...
- Le langage a une sémantique très précise.
- La taille des types primitifs est indépendante de la plate-forme.
- Java supporte un code source écrit en Unicode.
- Java est accompagné d'une librairie standard.

Java est distribué

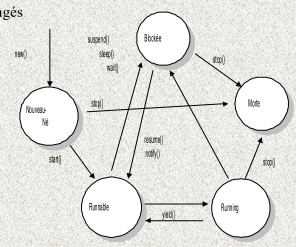
- API réseau (java.net.Socket, java.net.URL, ...).
- Chargement / génération de code dynamique.
- Applet.
- Servlet.
- Remote Method Invocation : RMI
- Interconnexion sur CORBA

Java est robuste

- A l'origine, c'est un langage pour les applications embarquées
- Gestion de la mémoire par un garbage collector (ramasse miettes)
- Pas d'accès direct à la mémoire
- Mécanisme d'exception
- compilateur contraignant (erreur si exception non gérée, si utilisation d'une variable non affectée, ...)

Java est multi-threadé

- Exécution de tâches en //
- Mémoire, Code et Ressources partagés
- Economie de ressources
- Un thread ~= méthode qui rend immédiatement la main
- Exemple événements (IHM, gc)
- + priorités
- + synchronisation
  - (moniteur, synchronized)
- Implantation dépendante du SE



Java est sécurisé

- Indispensable avec le code mobile.
- Pris en charge dans l'interpréteur.
- Trois couches de sécurité :
  - *Verifier* : vérifie le *byte code*.
  - *Class Loader* : responsable du chargement des classes.
  - *Security Manager* : accès aux ressources.
- Code certifié par une clé.

Java est semi-réflexif

- Le langage s'autodécrit
- Les éléments du langages sont pilotable de l'extérieur
- Possibilité de faire de la meta-programmation
- La machine virtuelle est pilotable de l'extérieur

API du JDK (Paquetages)

java.lang : classes de bases (+reflect)
java.io : entrées/sorties
java.util : utilitaires (structures, dates, events) (+zip)
java.net : réseau
java.applet : gestion des applets
java.awt : interface graphique (image, +datatransfert, +event)
java.beans : définition de composants réutilisables
java.math : entier de taille variable
java.rmi : invocation distante (+dgc, +registry, +server)
java.security : (+acl, +interfaces)
java.sql : jdbc ...
java.text : traduction, chaine=f(langue)

Architecture technique : Conclusion

- Notion de bac à sable
  - Isolation
  - Robustesse
  - En pleine évolution
- Utilise les ressources du système d'exploitation sous-jacent
  - Accès aux ressources (réseau, disque, mémoire...)
  - Mais …

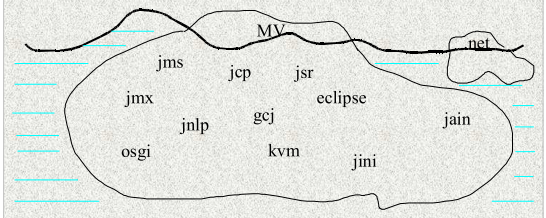
Le kit java

- JDK vs JRE
  - Java Development Kit / Java Runtime Environment
- Versions du jdk
  - 1.1 -> version de démonstration
  - 1.2 -> 1ère version utilisable (interfaces graphiques)
  - 1.3 -> 1ère version industrielle
  - 1.4 -> Version courante (compilation JIT)
  - 1.5 -> Version avancée (grande taille)
    - 40 Mo de librairies standards compressées
    - 13000 classes
    - 100k la machine virtuelle
- N'est pas open-source

La famille java

- **Java 2 SE**
  - java 2 standard edition (à partir du jdk1.4)
  - Kit et run-time standard java
- Java 2 ME
  - Java 2 Micro-Edition pour environnements contraints
  - CLDC / CDC
    - Connected Limited Device Configuration
    - Connected Device Configuration
  - Midp 2.0 : couche de spécialisation constructeur
- Java 2 EE
  - Java 2 Enterprise Edition
  - Principalement api EJB
  - Pas de machine virtuelle associée

L'Iceberg Java



eXtreme Programming

Stéphane Frénot

INSA Lyon - Département télécommunications Services et Usages

eXtreme programming Explored
William C. Wake
Addison Wesley, 2002

eXtreme programming : les concepts

- Séparation des rôles : client, programmeur, ...
- Couches clairement définies :
  - Programmation : comment écrire du code eXtreme
  - Travail d'équipe : comment organiser les équipes de développement
  - Processus : organisation du processus de développement

eXtreme prog. : développement

- Programmation
  - Conception simple
  - Approche par les tests
  - Refactorisation immédiate
  - Règles de codage

eXtreme programming : l'équipe

- Travail d'équipe
  - Propriété collective du code
  - Intégration continue du code
  - Utilisation de métaphores
  - Règles de codage
  - 40h / semaine
  - Programmation en tandem
  - Petites livraisons

eXtreme programming : processus

- Processus d'organisation
  - Client sur site
  - Approche par les tests
  - Petites livraisons
  - Jeux des prévisions

Le développement

le Développement : tester en premier

- Programmation incrémentale
  - petits codes
- Ecrire d'abord les tests
  - Le code est testable car il a été fait pour
  - Les tests sont testés : le test échoue si le code qui implante le test n'existe pas
  - Les tests sont répétables (car il s'agit de code)
  - Les tests aident à la documentation du code
  - Les tests forcent à définir le support minimal
- Existence de bibliothèque de test : JUnit (www.junit.org)
- http://www.xprogramming.com

Utilisation de JUnit

```
import junit.framework.*;
public class TestVector extends TestCase {
  public TestVector(String name){super(name);}
  public void testAddElement(){
    //Mise en place
    Vector v=new Vector();
    //Appels des méthodes
    v.addElement("Une chaine");
    v.addElement("Une autre chaine");
    //Assertion
    assertEquals(2, v.size());
  }
}
```

Cycle de codage eXtreme

- Ecrire un test

Cycle de codage eXtreme

- Ecrire un test
- Compiler le test il doit échouer

Cycle de codage eXtreme

- Ecrire un test
- Compiler le test il doit échouer
- Implanter le strict nécessaire

Cycle de codage eXtreme

- Ecrire un test
- Compiler le test il doit échouer
- Implanter le strict nécessaire
- Lancer le test et voir s'il échoue

Cycle de codage eXtreme

- Ecrire un test
- Compiler le test il doit échouer
- Implanter le strict nécessaire
- Lancer le test et voir s'il échoue
- Implanter le minimum pour réussir le test

Cycle de codage eXtreme

- Ecrire un test
- Compiler le test il doit échouer
- Implanter le strict nécessaire
- Lancer le test et voir s'il échoue
- Implanter le minimum pour réussir le test
- Refactoriser pour clarté et duplication

Cycle de codage eXtreme

- Ecrire un test
- Compiler le test il doit échouer
- Implanter le strict nécessaire
- Lancer le test et voir s'il échoue
- Implanter le minimum pour réussir le test
- Refactoriser pour clarté et duplication
- Recommencer au début

Cycle de codage eXtreme

- Ecrire un test
- Compiler le test il doit échouer
- Implanter le strict nécessaire
- Lancer le test et voir s'il échoue
- Implanter le minimum pour réussir le test
- Refactoriser pour clarté et duplication
- Recommencer au début
- Maintenir une TODO list

le Développement : test ?

Une question : Tout code peut il être testé ?
Exemple : difficulté d'automatiser le test d'une IHM

- Simulation de l'interface avec

  `getText, setText, doClick`
- Fabrication d'une version allégée du modèle

Q&A

- *Combien de temps dure un test ?*
  - De 1 à 5 minutes (10 au max)
- *Que faire si c'est plus long ?*
  - Réduire la taille du test
- *5 minutes ?*
  - Oui
- *L'A/R entre test et code fait une surcharge cognitive*
  - Non.. Vite
- *Ecrire un test ralentit le codage ?*
  - Non pas à terme

Réorganisation

Améliorer la conception du code sans modifier son comportement externe.

- Prérequis
  - code initial
  - tests unitaires
- Il faut :
  - une technique pour améliorer le code
  - un ensemble de réusinages à appliquer
  - un processus pour organiser le réunisage

Réorganisation : sentir le code

En regardant un classe, on peut "sentir" les réorganisations possibles.http://www.refactoring.com

- Classes trop grandes

Réorganisation : sentir le code

En regardant un classe, on peut "sentir" les réorganisations possibles.http://www.refactoring.com

- Classes trop grandes
- Méthodes trop grandes

Réorganisation : sentir le code

En regardant un classe, on peut "sentir" les réorganisations possibles.http://www.refactoring.com

- Classes trop grandes
- Méthodes trop grandes
- Switch (à la place du polymorphisme)

Réorganisation : sentir le code

En regardant un classe, on peut "sentir" les réorganisations possibles.http://www.refactoring.com

- Classes trop grandes
- Méthodes trop grandes
- Switch (à la place du polymorphisme)
- Classe de structure (get/set sans méthodes)

Réorganisation : sentir le code

En regardant un classe, on peut "sentir" les réorganisations possibles.http://www.refactoring.com

- Classes trop grandes
- Méthodes trop grandes
- Switch (à la place du polymorphisme)
- Classe de structure (get/set sans méthodes)
- Code dupliqué

Réorganisation : sentir le code

En regardant un classe, on peut "sentir" les réorganisations possibles.http://www.refactoring.com

- Classes trop grandes
- Méthodes trop grandes
- Switch (à la place du polymorphisme)
- Classe de structure (get/set sans méthodes)
- Code dupliqué
- Code dupliqué à un delta près

Réorganisation : sentir le code

En regardant un classe, on peut "sentir" les réorganisations possibles.http://www.refactoring.com

- Classes trop grandes
- Méthodes trop grandes
- Switch (à la place du polymorphisme)
- Classe de structure (get/set sans méthodes)
- Code dupliqué
- Code dupliqué à un delta près
- Surdépendance de types de base

Réorganisation : sentir le code

En regardant un classe, on peut "sentir" les réorganisations possibles.http://www.refactoring.com

- Classes trop grandes
- Méthodes trop grandes
- Switch (à la place du polymorphisme)
- Classe de structure (get/set sans méthodes)
- Code dupliqué
- Code dupliqué à un delta près
- Surdépendance de types de base
- Commentaires inutiles

Quand arrêter la refactorisation ?

On arrête la refactorisation quand le programme :

- 1) Réussit les tests
- 2) Communique tout ce qu'il doit communiquer
- 3) N'a pas de duplication
- 4) Possède un minimum de classes et de méthodes

==> Once and Only Once
==> DRY : Don't Repeat Yourself

(The pragmatic programmmer)

Travail d'équipe

Travail d'équipe

- Propriété du code

Travail d'équipe

- Propriété du code
- Intégration du code

Travail d'équipe

- Propriété du code
- Intégration du code
- Surcharge de travail

Travail d'équipe

- Propriété du code
- Intégration du code
- Surcharge de travail
- Espace de travail

Travail d'équipe

- Propriété du code
- Intégration du code
- Surcharge de travail
- Espace de travail
- Plannification des livraisons

Travail d'équipe

- Propriété du code
- Intégration du code
- Surcharge de travail
- Espace de travail
- Plannification des livraisons
- Standard de codage

Propriété du code

A qui appartient le code ?

- Personne : perdu dans les méandres du temps

Propriété du code

A qui appartient le code ?

- Personne : perdu dans les méandres du temps
- Dernier à l'avoir touché / Le nouveau (marque, chaises musicale / test par le feu)

Propriété du code

A qui appartient le code ?

- Personne : perdu dans les méandres du temps
- Dernier à l'avoir touché / Le nouveau (marque, chaises musicale / test par le feu)
- Un propriétaire par classe/package (Monogamie) : on ne touche pas à MON code

Propriété du code

A qui appartient le code ?

- Personne : perdu dans les méandres du temps
- Dernier à l'avoir touché / Le nouveau (marque, chaises musicale / test par le feu)
- Un propriétaire par classe/package (Monogamie) : on ne touche pas à MON code
- Sequential owner (Propriété locative / Serial Monogamy) : problème du long terme

Propriété du code

A qui appartient le code ?

- Personne : perdu dans les méandres du temps
- Dernier à l'avoir touché / Le nouveau (marque, chaises musicale / test par le feu)
- Un propriétaire par classe/package (Monogamie) : on ne touche pas à MON code
- Sequential owner (Propriété locative / Serial Monogamy) : problème du long terme
- Propriété par couche (tribalisme) (client/serveur) Réduit la "Bus number" law

Propriété du code

A qui appartient le code ?

- Personne : perdu dans les méandres du temps
- Dernier à l'avoir touché / Le nouveau (marque, chaises musicale / test par le feu)
- Un propriétaire par classe/package (Monogamie) : on ne touche pas à MON code
- Sequential owner (Propriété locative / Serial Monogamy) : problème du long terme
- Propriété par couche (tribalisme) (client/serveur) Réduit la "Bus number" law
- Propriété collective

Propriété collective

Quels sont les problèmes potentiels

- Fierté du code
- Tragedie des communs (tout le monde est responsable, donc personne)
- Expertise
- Mic/Mac de code (mélange de style)
- Marcher sur les "pieds" de l'autre (Mise en attente du développement)

Propriété collective : Xp solutions

Quels sont les problèmes potentiels

- Fierté du code : équipe Xp
- Tragedie des communs : tandem + test
- Expertise : deep expertise
- Mic/Mac de code : standard de code
- Marcher sur les "pieds" de l'autre : Intégration continue

Quelques règles de fonctionnement 1

Quand intégrer le code ?

- Juste avant la livraison : Spécial WE difficiles
- 1 fois par jour : le soir (et si ca ne marche pas...)
- Intégration en continue : Approche XP

Quelques règles de fonctionnement 1

Quand intégrer le code ?
- Juste avant la livraison : Spécial WE difficiles
- 1 fois par jour : le soir (et si ca ne marche pas...)
- Intégration en continue : Approche XP

Comment travailler ?
- Finir coûte que coûte (death march)
- 40 heures / semaine max. (+5 heures de veille techno)

Quelques règles de fonctionnement 2

Comment organiser l'espace de travail ?
- Séparation géographique (Réunion trimestrielle)
- En office (1 à 2 personnes)
- Cubicle
- Bureaux paysagiste

Quelques règles de fonctionnement 2

Comment organiser l'espace de travail ?
- Séparation géographique (Réunion trimestrielle)
- En office (1 à 2 personnes)
- Cubicle
- Bureaux paysagiste

Organisation des livraisons (releases) ?
- petites livraisons
- livraison ZFR (Zero Feature Release)

Quelques règles de fonctionnement 3

Quels codes écrire ?

```
for(;;){                | while(true){}
------------------------+---------------------------
if(i==j)                | if(i==j){
{                       |   System.out.println();
  System.out.println(); | }
}                       |
------------------------+---------------------------
if(i==j)                | if(i==j){
  System.out.println(); |   System.out.println();
                        | }
```


Quelques règles de fonctionnement 4

Quels standards choisir
- Aucun : ARRRGH
- Choix du programmeur (création) et s'impose à l'ajout
- Choix du programmeur (création) et se plie à l'ajout
- Choix d'équipe : exemple microsoft, javabeans ...

Quelques règles de fonctionnement 4

Quels standards choisir
- Aucun : ARRRGH
- Choix du programmeur (création) et s'impose à l'ajout
- Choix du programmeur (création) et se plie à l'ajout
- Choix d'équipe : exemple microsoft, javabeans ...

JavaBeans :
- 4 espaces d'indentation
- Ouverture des accolades sur la même ligne
- Convention de nommage des javabeans (get, set, is...)

Développement en Tandem

Comment organiser le développement
- Solo : Codage tout seul dans un coin
- Equipe entière : Tout le monde participe
- Désengagement : Le partenaire regarde l'air distrait

Développement en Tandem

Comment organiser le développement
- Solo : Codage tout seul dans un coin
- Equipe entière : Tout le monde participe
- Désengagement : Le partenaire regarde l'air distrait
- Programmation par paire
  - changement régulier de clavier
  - changement régulier de partenaire
  - propriétaire de la tâche (un des deux)
  - difficulté de trouver des paires
  - junior/senior
  - 1/2 productivité

Tandem : une analyse

- Le partenaire demande de l'aide au moment de la formation de l'équipe
- Le partenaire aide aussi bien globalement que techniquement
- Il existe un protocole d'échange de clavier
- La paire apprend ensemble; l'apprentissage est dans le processus
- Le partenaire offre une vision externe de qualité
- Le partenaire fournit l'autorisation (l'excuse) de faire (ou ne pas faire) une chose
- Si un partenaire oubli quelque chose, l'autre le remet à l'ordre

Y a t'il une "architecture" ?

- Approche orientée architecture
  - *"Certaines choses sont difficiles à changer, donc il faut concevoir un squelette avant"*
- Approche orientée Xp
  - *"Accepter de s'adapter au changement (merci l'objet)"*

L'Architecture Xp

- Principe d'exploration
- Principe de la métaphore
- La première itération (squelette)
  - Faire la surface du programme (hello, world)
  - Tout configurer
  - Supprimer les fonctionnalités
  - Interface minimaliste
- Petites livraisons
- Refactorisation
- Pratique d'équipe

Q&A

- Qui définit l'architecture de déploiement ?
  - L'équipe au moment de l'exploration
- Comment est documentée l'architecture ?
  - Test, plus ce que veut le client... (coût)
- La version de surface (ou réaliser la persistance ?)
  - Permet au client de commencer à tester
- Si tout le monde est architecte alors personne ?
  - Notion de coach
- Donc on utilise un architecte ?
  - Non, il est plus important de l'intégrer dans Xp
- Où ca va planter ?
  - MCD et MultiThread

Principe de la métaphore

Pourquoi chercher une métaphore ?

- Vision commune
- Partage du vocabulaire
- Créativité
- Identifie une architecture

Principe de la métaphore

Pourquoi chercher une métaphore ?

- Vision commune
- Partage du vocabulaire
- Créativité
- Identifie une architecture

Quelques exemples de métaphores

- Le bureau pour les interfaces utilisateurs
- Ligne de montage pour la gestion de flux
- Caddie électronique pour le e-commerce
- L'annuaire papier pour l'annuaire
- Le courrier papier pour la messagerie

Utilisation de la métaphore

- Choisie pendant la phase d'exploration
- Remise en question au fil de l'eau
- Elle guide la solution
- Utiliser ses concepts pour les classes principales

Utilisation de la métaphore

- Choisie pendant la phase d'exploration
- Remise en question au fil de l'eau
- Elle guide la solution
- Utiliser ses concepts pour les classes principales

Limites de la métaphore

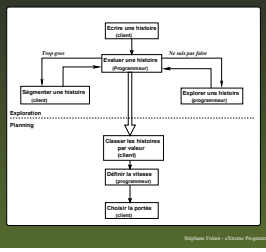
- Non familière à tout le monde
- La métaphore peut être trop faible
- La métaphore peut être trop forte
- La métaphore peut borner la conception
- Attention aux mots "magique", "super"

Les processus

Processus

- Plannification de release (sorties)
- Organisation d'une itération
- Les acteurs

Plannification d'une release



Release : Exploration

- But :
  - Comprendre le but du système afin de pouvoir estimer la charge
- Méthode :
  - Le client écrit une histoire, le programmeur l'évalue
- Résultat :
  - Ensemble d'histoires évaluées
- Durée :
  - Jour ==> Semaines

Les rôles

- Client écrit une histoire : testable, 1 à 3 "semaines"
- Programmeur évalue l'histoire
- Programmeur fait une exploration (étayer)
- Client découpe une histoire pour simplifier
- A la fin : chaque histoire à un coût

==> Notion de semaine "idéale"

Release : Planning

- But :
  - Plannifier la prochaine livraison
- Entrées :
  - Des histoires
- Méthode :
  - cf. transparents suivants
- Résultat :
  - Une liste triée par priorité des histoires à livrer la prochaine fois
- Durée :
  - Quelques heures

Planning : comment

- Client trie les histoires par valeur (haute, moyen, basse)(doit, peut,pourrait)
- Programmeur déclare la vitesse
  - nombre de points / itération
  - 1ère itération = 1 à 3 semaines
  - estimation : 1/3 d'histoire / programmeur / s.
  - après : la règle du "temps d'hier"
- Client sélectionne la portée
  - Le client sélectionne les histoires
  - La durée = nb points / vitesse
  - La première : L'ensemble du système end<->end

Q&A

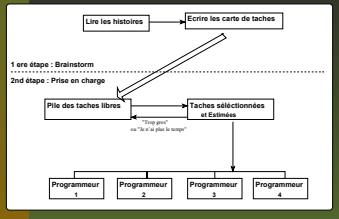
- Si le client n'aime pas le résultat, il peut :
  - changer les histoires, livrer moins d'histoires, accepter une nouvelle date, changer une partie de l'équipe, quitter le projet
  - il ne peut pas changer les estimations
- Le client est il lié aux priorités fixées ?
  - Non, c'est un point de départ
- Peut on trier par risque ?
  - Why not, mais on peut réduire les risques en diminuant la taille des histoires

"Software developpement as Cooperative Game"
http://members.aol.com/humansandt/papers/asgame/

Exemple : Vitesse 5 (5 / 3 semaines)

high	Z39.50 (3)	3	Z39.50
	Query (2)	2	Query
	MARC (2)	----	----
	GUI (2)	2	MARC
	Performance (1)	2	GUI
Medium	Boolean (3)	1	Performance
	Sorting (2)	----	----
	Drill-Down (2)	1	SUTRS
	SUTRS (2)	2	Config
	Config (2)	1	Portable
	Portable (1)		
low	Print (2)		
	No-Drill (1)		
	Save Result (1)		
	Save Query (1)		

Plannification d'une release



Mise en place du jeu

- Le suiveur (trackeur) indique le nombre d'histoires réalisées la fois précédente
- Le client sélectionne les tâches inacomplies < limite des points
- Le tracker donne pour chaque programmeur ses points de programmation (1ère : 0.5 /programme/jours de l'iteration)
- Chaque programmeur reçoit son nombre de points (yesterday's rule)

Préparation d'une itération

Phase 1 : Brainstorm d'équipe
- Le client sélectionne une des histoires et la lit
- Les programmeurs définissent les tâches
- Le processus est répété pour chaque histoire

Préparation d'une itération

Phase 1 : Brainstorm d'équipe
- Le client sélectionne une des histoires et la lit
- Les programmeurs définissent les tâches
- Le processus est répété pour chaque histoire

Phase 2 : Les programmeurs
- Pour chaque tâche un programmeur :
  - la sélectionne
  - Estime le temps qu'il passera dessus (en points programme)
  - Ecrit ce temps sur la carte
- Si la tâche est trop grosse (> 3 jours) le client décompose la tâche

Fin d'itération

- Pas de solution
  - Le client choisit une histoire à décomposer ou à remettre à plus tard
- L'équipe a besoin de plus d'histoires
- L'équipe gagne : Tout le monde est à peu prêt à 0

A la fin l'équipe a un plan d'itération des histoires
- Le gestionnaire enregistre le plan
- Le client écrit les tests de validation
- Les programmeurs bossent !

Q& A

- Durée : Quelques heures
- Pourquoi des tâches et pas des histoires ? plus simple à estimer
- Chaque programmeur choisit sa tâche ==> pas de guru
- Qui possède la tâche programmeur / tandem ?
- Pourquoi ce n'est pas l'équipe qui évalue les tâches ? Implication du programmeur
- Somme des tâches != histoire ? Les tests valident l'histoire
- Estimation ? le passé + en faisant des tests de 5 - 60 minutes
- Estimation en fonction du partenaire ? oui

Les rôles

- Le client
- Le programmeur
- Les gestionnaires
  - Le coach
  - Le trackeur
  - Le mentor

Le client

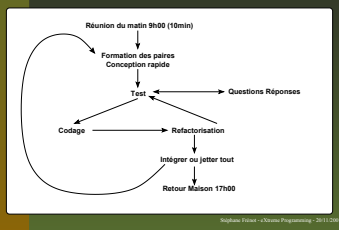
- Client : 4 rôles pendant l'itération + 1 pendant la recette
  - 1) Répondre aux questions
  - 2) Ecrire les tests
  - 3) Lancer les tests
  - 4) Piloter l'itération
    - Sélectionne le contenu d'une release
    - Plannifie les itérations
    - Prend des décisions au vol
  - 5) Valide la livraison (c'est la fête)

Q& A

Les problèmes du client
- J'ai un boulot à côté :
  - il faut dégager du temps
- Je ne peux pas indiquer aux analystes... :
  - non, il n'y en a pas
- Que faire s'il y a plein de clients :
  - il faut sérialiser les clients
  - Le système est fondé sur un seul "flux" client
- Mais vraiment plein, plein de clients :
  - il faut s'adapter

Les programmeurs

Teste, code, refactorisation (!= conception, code, test)



Le manager

- Représentant du projet au monde extérieur
- Forme l'équipe
- Obtient les ressources
- Pilote l'équipe
  - Présente les avancements
  - Convoque les réunions
  - Organise les célébrations
- Gère les problèmes (client, membres...)

Le manager ne donne

- Pas de priorités (client)
- Pas d'assignation de tâches (programmeur)
- Pas d'estimations (programmeur)
- Pas de calendriers (négociation client/programmeur)

==> Masque aux programmeurs les problèmes

Le tracker et le coach

Le tracker

- Suit le planning de livraison (histoires)
- Suit le planning d'iteration (tâches)
- Suit l'état de validation des tests

Le coach

- Attributs : droit, respecté et respecte, sur-site, souvent excellent programmeur
- Activités : surveille les processus (réunions qui débordent, etc...), maintient les règles, modifie le processus, mentor, fournit les distractions

conclusions

Les problèmes à surveiller

- Ralentissement de la vitesse (refactoring, test ?)
- Mauvais code ou usine à gaz (refactoring)
- Baisse de la qualité (tests)

conclusion

Quels sont les points forts d'XP ?

- Communication rapide et efficace (client, programmeur)
- Validation primordiale (réduction du temps, car tout est validé)
- Le principe d'itération fonctionne (iter. 1 à 3 s., rel. qq mois)

conclusion

Quels sont les points forts d'XP ?

- Communication rapide et efficace (client, programmeur)
- Validation primordiale (réduction du temps, car tout est validé)
- Le principe d'itération fonctionne (iter. 1 à 3 s., rel. qq mois)

Quels sont les points à améliorer ?

- Simplification et éclaircissement (processus de plannification)
- Limites d'Xp : Grosse équipe ? Autres domaines ? Quand ?
- Trouver des concepts fondateurs
- Tests d'acceptation : validation dans d'autres contextes
- Rôle du client : Xp se focalise surtout sur le programmeur

Bibliographie

1. Hunt, Andrew and D. Thomas. 2000 – The pragmatic programmeur : From Journeyman to Master. Boston : Addison-Wesley
2. . 2000 – Programming Pearls, Second Edition. Boston : Addison-Wesley
3. 2000-2001 – xp series – Addison-Wesley

Sites web

1. http://www.egroups.com/group/extremeprogramming
2. http://www.extremeprogramming.org
3. http://www.junit.org
4. http://www.pairprogramming.com
5. http://www.refactoring.com
6. http://www.xpdeveloppeur.com
7. http://www.xprogramming.com
8. http://www.xp123.com

Département
Télécommunications
Services & usages

PROJETS PI

2007

stephane.frenot@insa-lyon.fr

Résumé

L'objectif des Projets Innovations (PI) est de permettre aux étudiants du département Télécommunications de monter, de suivre et de présenter un projet qu'ils auront réalisé durant le second semestre de la 4ème année. Les projets innovation sont des projets entièrement portés par l'équipe de projet qui en définit l'objectif initial (la Demande) et en fait une offre cohérente présentée à la fin.

Le projet est intégralement décrit sur `http://tc-net2.insa-lyon.fr/lutece/jsp/site/Portal.jsp?page_id=36`

Les projets innovation reposent sur deux axes :

- un cours de formation à la conduite de projets et aux outils associés,
- des heures projets suivies par un binôme enseignant technique / enseignant humanité qui supervise la conduite du projet

L'évaluation finale se fait principalement sur le bon déroulement du projet (respect des livrables, organisation de l'équipe...) mais ni sur une "bonne réalisation technique", ni sur une "bonne soutenance finale".

Table des matières

1 La formation associée

Le cours repose sur les prérequis de 3TC dans le cadre du cours de Management de projets. Les enseignements pour PI sont répartis de la manière suivante.

- 2h Cours Analyse fonctionnelle
- 2h Gestion de projets et GANTT
- 2h Méthodes de management
- 2h Projets Agiles
- 2h Brevets

L'ensemble des supports de cours correspondant est disponible sur `http://` `http://tc-net2.insa-lyon.fr/lutece/jsp/site/Portal.jsp?page_id=36`

2 Les projets

2.1 Présentation

Les projets sont proposés, mis en oeuvre et défendus par les étudiants. Nous faisons une exception sur des projets proposés par des entreprises externes. Les projets sont structurés en deux phases. La première phase est une phase de proposition OSEO-ANVAR où l'équipe dépose son projet et le découpage associé. La seconde phase est la phase de mise en oeuvre où les projets sont suivis par un binôme d'enseignants qui veille au bon déroulement du projet. A la fin de cette seconde phase, une note est établie qui est la note plafond du projet. Une soutenance du projet est faite en fin d'année.

2.2 Déroulement

2.2.1 Avant projet

Au premier semestre les étudiants sont informés de la nouvelle formule. Ils se constituent en équipes et commencent à rechercher des idées. Les idées proviennent des étudiants eux-mêmes, des cours de 3 et 4 TC. Les enseignants et autres peuvent proposer des idées (un paragraphe) sur un site Web. Ces idées sont offertes aux étudiants qui se l'approprient intégralement. Concernant les projets apportés par des entreprises externes, les projets sont proposés comme les autres, mais l'entreprise restera le client du projet (on retombe alors dans le même mode de fonctionnement que les PTS cf. plus loin). Si plusieurs équipes veulent faire les projets proposés par l'extérieur, le responsable des PI tranche pour une équipe. Cette partie de réflexion a lieue avant le lancement réel des projets au second semestre.

2.2.2 Choix des sujets (2 semaines)

- Les étudiants sont constitués en équipes et choisissent un sujet. Le sujet provient soit d'eux mêmes, soit de la "boite à idées", soit des extérieurs. Le dépôt de sujet se fait en 1 page pdf, qui décrit le projet, la liste des membres de l'équipe (mail). Un exemple de proposition est fourni en annexe.

2.2.3 Cahier OSEO-ANVAR (1,5 mois)

Le dossier Oseo est une proposition qui doit permettre de comprendre le projet. Ce dossier ne s ?adresse pas qu ?à des spécialistes techniques du domaine visé. Il doit être convaincant sur trois axes.

- Le caractère innovant du projet ou l ?intérêt de l ?innovation proposée : sa différence par rapport à l ?existant, son utilité technique, économique, sociale, etc.
- L'expertise technique doit permettre de comprendre les grandes lignes de l'architecture visée.
- L'expertise économique doit rendre compte de la faisabilité économique du projet (cet aspect n ?est pas abordé dans la phase II).

L'étude de marché doit vérifier l'existence d'une véritable opportunité de se lancer. L'Agence pour la création d'entreprises (http ://www.apce.com) note que l'étude de marché permet :

- De mieux connaître les grandes tendances et les acteurs d'un marché visé.
- De réunir suffisamment d'informations permettant de fixer des hypothèses de chiffre d'affaires.
- De déterminer sa stratégie et son "marketing mix".
- De fixer un budget prévisionnel.

L'étude de marché doit clairement identifier le marché visé (professionnels/particuliers, stagnation/développement, faiblement/fortement concurrentiel, etc.), les concurrents (leur identité, leur taille, les agressivité, leurs méthodes de distribution, leurs tarification, etc.) et le cadre législatif (certifications, licences, règles de santé publique spécifiques, etc.). Seule une telle étude peut permettre de déterminer la plausibilité du projet.

Le dossier Oseo doit également contenir une annexe importante concernant le déroulement de la seconde phase. Cette annexe doit permettre de comprendre le contour fonctionnel du prototype fourni durant la suite du projet. Cette annexe doit contenir :

- la liste des grandes fonctions du prototype
- un lotissement de la phase II
- une description et un chiffrage du matériel nécessaire au prototype

Le cahier est limité à 25 pages. Un exemple partiel de cahier est fourni en annexe.

L'évaluation des cahier se fait par un ensemble d'experts techniques. Il y a au moins 2 experts par projet et le document de retour d'expertise est fourni en annexe. A la suite de ces évaluations les tuteurs de projets remontent également leurs commentaires sur chacun des projets. Il est possible alors d'arrêter un projet pour diverses raisons (faiblesse du dossier, impossibilité technologique...).

L'évaluation du cahier Oseo et du déroulement de la phase I du projet donne 1/3 de la note finale de projet.

A la suite de cette première phase les projets continuent avec de nouveaux tuteurs enseignants et humanité.

2.2.4 Réalisation du projet (3,5 mois)

Dans cette seconde phase les projets sont plus ou moins autonomes. Les tuteurs voient les projets environ 6 fois durant cette période. L'objectif des réunion est de contrôler l'avancement du projet, le respect des délais, cahier de charges et budget. Pendant cette phase on demande aux étudiants de déposer leur do-

cumentation officielle sur un site de dépôt et de réaliser un site de présentation externe de leur projet/entreprise sous un outil de gestion de contenu (CMS).

A la fin de cette phase une réunion de recette est faite avec les tuteurs. A ce moment le projet est évalué une note est remontée par les encadrants qui correspond au 2/3 restants de la note finale.

2.3 Finalisation

2.4 Présentation finale

L'ensemble des projets est présenté devant tous les étudiants, les enseignants tuteurs et toutes les personnes du département qui sont intéressées par ces projets. Les soutenances sont classiquement faites en 1 journée ou 2 demi-journées. A la suite des présentation une sélection des trois meilleurs projets est faite. Cette sélection porte principalement sur la soutenance. Toutefois la soutenance ne conduisent pas à une modification de la note de projet. Les trois projets retenus sont sélectionnés pour être présentés lors d'un forum partenaires dans les jours suivants.

2.5 Rendus et pérennisation des projets

Les projets représentent un grand volume d'activité de la part des étudiants et du département. Les projets sont donc conservés au département sous différentes formes :

- L'intégralité des documents 'officiels' ; proposition initiale, cahier Oseo, cahier de recette et codes sources du prototypes sont conservés en ligne sur un site Web
- Tous les projets maintiennent un site Web privé à l'INSA qui reste en ligne. Une liste des sites en consultation est disponible sur `http://tc-net2.insa-lyon.fr`
- Chaque projet fournit un poster de synthèse de son projet. Ce poster A1 est imprimé et utilisé dans le cadre du forum. Un exemple de poster de projet est donné en annexe.

Parmi les projets sélectionné nous avons déclenché la première phase de recherche d'antériorité pour brevet. Cette recherche se fait conjointement avec INSAValor et permettrait de libérer les droits pour une éventuelle industrialisation d'un projet.

2.6 Synthèse sur la notation

```
* 1ère partie : Dossier OSEO
  50 % Structuration du dossier
   - Mise en page (plan, annexes, synthèse, bibliographie, page, décomposition,
     polices, lisibilité) : Donner des pistes pour la rédaction

  50 % Contenu
   - Décomposition en Tâches (Durées, contenu, réalisme)
   - Décomposition en fonction (description, réalisme)
   - Idée nouvelle
   - Transversalité (touche plusieurs axes
     informatique/réseau/télécommunication)

==> Retour : un rapport d'expertise sur la structure et le contenu. Le contenu
est évalué par un expert, qui analyse plusieurs dossiers.

* 2nd partie : Déroulement du projet
  40 % suivi des projets
    - respect des délais
    - réorganisation des délais
    - gestion de l'équipe
  30 % Fonctionnement de l'équipe
    - ambiance de l'équipe
    - respect des relations
    - homogénéité du travail (répartition de tâches)

  20 % réalisation technique (maquette / présentation)
  10 % rapport technique

==> Retour : un rapport d'évaluation technique. 1 seule expertise

* la soutenance n'intervient que pour conforter la note :)
```

3 Les moyens

3.1 Encadrements

Les projets sont suivis par des binômes TC/Huma, à raison de 3 équipes par binôme. L'encadrement des trois équipes est de 10h par projet. Le binôme voit l'équipe projet environ 3 fois pour la première phase et 5 fois pour la seconde.

Les équipes possèdent un chéquier de 15h d'expertise technique et prennent RDV avec les enseignants concernés. Il est demandé que toutes les équipes aient au moins vu 1 expert extérieur qui garanti certain éléments techniques. Les experts peuvent être consultés aussi bien en phase 1 qu'en phase 2.

Le bilan en heures cours/encadrement/expertise est de :

- 14h eqTD pour les cours
- 200h eqTD d'encadrement de projet
- 34h eqTD d'expertise (cahier Oseo et expertise de projet)

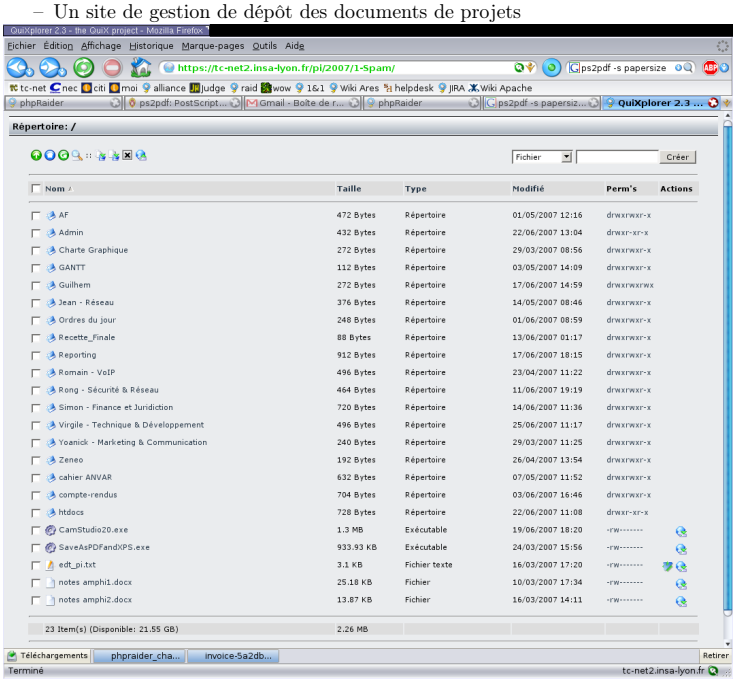
3.2 Ressources

3.2.1 Sites Webs

Le département fournis les éléments suivant pour la gestion de projet.

– Un site Web de centralisation de l'information `http://tc-net2.insa-lyon.fr/lutece/jsp/site/Portal.jsp?page_id=36`

– Un site de gestion de dépôt des documents de projets



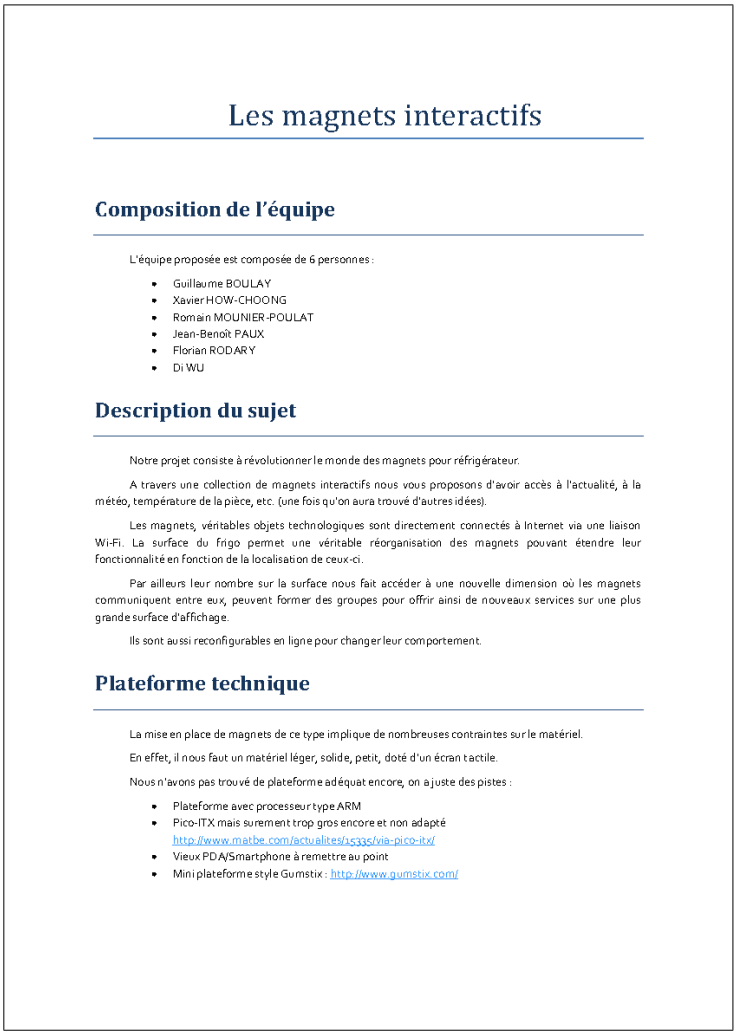
– Un site par projet pour la gestion externe du projet (maintenu par les étudiants)

3.2.2 Ressources

Nous fournissons les documents suivants comme support aux projets :
– Conduite de réunion, comment organiser/se comporter en réunion
– Rédaction d'un rapport, quels sont les éléments que l'on doit retrouver dans un rapport
– Normes et références pour la rédaction de dossier Oseo
– Normes et références pour la rédaction de cahier des charges

A Annexes

A.1 Proposition initiale



Les magnets interactifs

Composition de l'équipe

L'équipe proposée est composée de 6 personnes :

- Guillaume BOULAY
- Xavier HOW-CHOONG
- Romain MOUNIER-POULAT
- Jean-Benoît PAUX
- Florian RODARY
- Di WU

Description du sujet

Notre projet consiste à révolutionner le monde des magnets pour réfrigérateur.

A travers une collection de magnets interactifs nous vous proposons d'avoir accès à l'actualité, à la météo, température de la pièce, etc. (une fois qu'on aura trouvé d'autres idées).

Les magnets, véritables objets technologiques sont directement connectés à Internet via une liaison Wi-Fi. La surface du frigo permet une véritable réorganisation des magnets pouvant étendre leur fonctionnalité en fonction de la localisation de ceux-ci.

Par ailleurs leur nombre sur la surface nous fait accéder à une nouvelle dimension où les magnets communiquent entre eux, peuvent former des groupes pour offrir ainsi de nouveaux services sur une plus grande surface d'affichage.

Ils sont aussi reconfigurables en ligne pour changer leur comportement.

Plateforme technique

La mise en place de magnets de ce type implique de nombreuses contraintes sur le matériel.

En effet, il nous faut un matériel léger, solide, petit, doté d'un écran tactile.

Nous n'avons pas trouvé de plateforme adéquat encore, on ajuste des pistes :

- Plateforme avec processeur type ARM
- Pico-ITX mais surement trop gros encore et non adapté
  http://www.matbe.com/actualites/15335/via-pico-itx/
- Vieux PDA/Smartphone à remettre au point
- Mini plateforme style Gumstix : http://www.gumstix.com/

A.2 Cahier Oseo

SPITSink

En partenariat avec :

INSA LYON — nextiraOne — SPIE

Informations sur le document :

Référence du projet	PI-Groupe 1
Equipe	Zeneo
Titre du projet	SPITSink
Intitulé du sujet	Solution Anti-SPIT
Tuteurs	David Gindis, Isabelle Auge-Blum
Date de remise finale	30 Mars 2007
Nom du délivrable	Cahier ANVAR
Version du document	2.0
Chef de projet	Guilhem TESSEYRE
Interface du projet	https://tc-net2.insa-lyon.fr/1-Spam/
Mots-Clés	VoIP, Spam, Security, IPBX, Telephony
Site web du projet	http://zeneo.insa-lyon.fr

Historique du document

Version	Date	Commentaires
Draft	16/03/07	Première ébauche de l'ANVAR
1.0	25/03/07	Première version avec l'essentiel du contenu
1.1	29/03/07	Intégration d'une charte graphique de meilleure qualité Ajout de l'historique du document Ajout de la liste des figures et des tables Ajout du slogan Réduction de la présentation de l'équipe
1.2	29/03/07	Ajout des correctifs sur la partie financière Ajout d'un scénario minimum dans la partie financière
1.3	29/03/07	Ajout des correctifs sur la partie technique
1.4	29/03/07	Ajout des correctifs sur la partie marketing
1.5	29/03/07	Refonte du résumé exécutif et de la présentation du contexte Numérotation depuis le résumé exécutif Redéfinition des tâches du GANTT Mise en page générale
1.6	30/03/07	Intégration des derniers correctifs Rectification de la partie financière Mise en place des références dans le texte Mise en page générale
2.0	30/03/07	Relecture et correction orthographique Version Finale

SPITSink

Sommaire

ZENEO

Liste des tableaux

Liste des figures

SPITSink

Abréviations

CA	Chiffre d'Affaire
DoS	Deny Of Service
GE	Grandes Entreprises
GSM	Global System for Mobile Communication
INPI	Institut National de la Protection Intellectuelle
IP	Internet Protocol
IP-PBX	Internet Protocol-based Private Branch Exchange
IPBX	Idem IP-PBX
PME	Petites et Moyennes Entreprises
RTC	Réseau Téléphonique Commuté
SIP	Session Initialisation Protocol
SPIT	Spam Over Internet Telephony
TPE	Très Petites Entreprises
ToIP	Telephony Over Internet Protocol
VoIP	Voice Over Internet Protocol
VPN	Virtual Private Network

Références

VoIP & SPIT:
[1] France Telecom,R&D, http://www.iptel.org/voipsecurity/doc/08%20-%20Mathieu%20-%20SPIT%20Mitigation%20by%20a%20Networ-Level%20Anti-Spit%20Entity.pdf
Méthodologie sur l'implémentation d'un module anti-SPIT et analyse des besoins, *Juin 2006.*

[2] Sun Microsystems, http://www.java.sun.com/products/jain/JAIN-SIP-Tutorial.pdf
Explications sur l'utilisation de JAIN SIP, pour l'implémentation de modules pour la ToIP, *2003.*

[3] Network Working Group, www.javvin.com/protocol/rfc3261.pdf
Draft du protocole SIP. Documentation technique complète sur la technologie de base de la ToIP.*Juin 2002*

[4] Ram Dantu, Prakash Kolan, www.usenix.org/events/sruti05/tech/full_papers/dantu/dantu.pdf
"Detecting Spam in VoIP Networks". Explications de différents algorithmes de détection des SPIT, *2005*

[5] ToIP Security, présentation de Jean-Pierre Kellermann, Alcatel-Lucent Enterprise Forum, *Février 2007*

Sources marché :
[6] http://www.ilotech.com/article.php?sid=865
Etude du marché de la téléphonie sur IP d'après diverses études (Solucom, Pouey, etc.). *03/2006*

[7] http://www.solucom.fr/communiques.php?CLIENT=ACTUS-0-121&ID=ACTUS-0-6103
Etude réalisée par Solucom auprès d'un échantillon représentatif d'entreprises françaises. *Janvier 2006*

Concurrence :
[8] NEC Seal: http://www.generation-nt.com/actualites/23595/nec-voip-seal-protection-spit-spam-voip
News écrite par Christian D. sur le site generation-net.com. Il y décrit le système de NEC de lutte contre le SPIT. *27/01/2007*

[9] Eyeball Anti-SPIT: http://www.eyeball.com/products/anti_spit_server.html
Description du produit, Anti-SPIT, sur le site du constructeur, Eyeball. *2005*

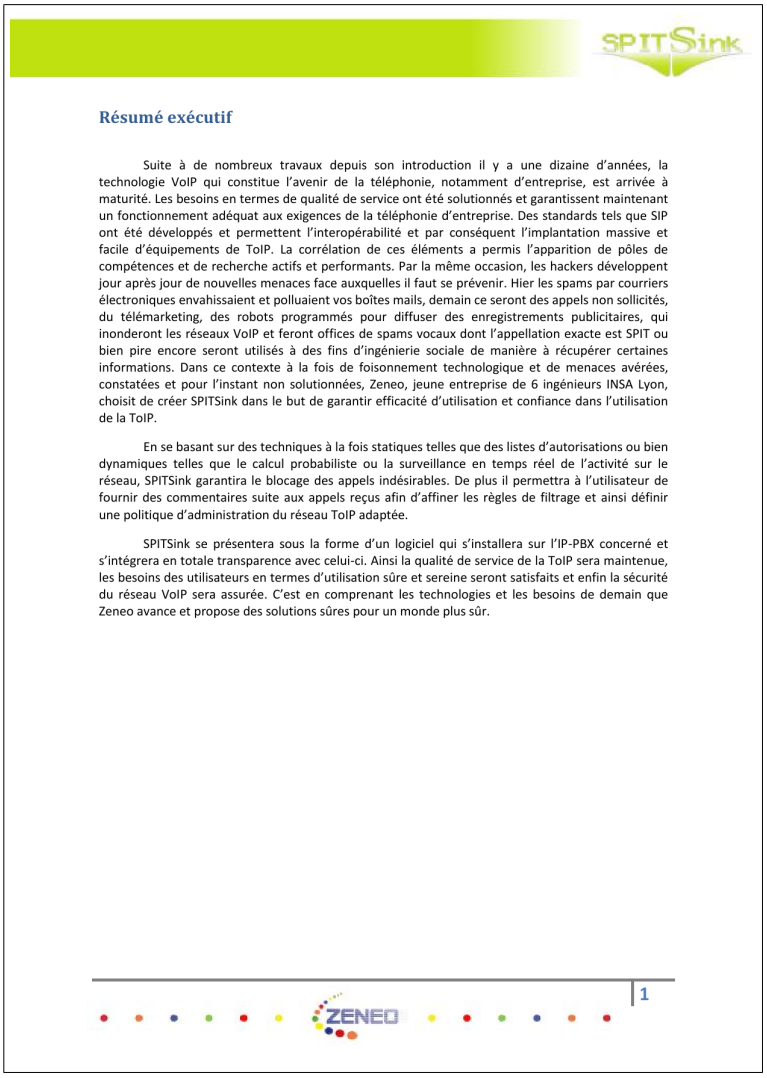
[10] Qovia: http://www.networkworld.com/news/2004/071204qovia.html
Article écrit par Cara Garretson sur le site NetworkWorld. Elle y décrit le système de Qovia de lutte contre le SPIT et le marché potentiel de l'époque. *12/07/04*

[11] http://networks.silicon.com/telecoms/0,39024659,39165534,00.htm?r=4
Is Spit set to soar? Par Richard Thurston. News présentant la concurrence : NEC, Eyeball et le fait que Cisco ne ressente la menace du SPIT.

[12] Grand Central: http://www.lexpansion.com/art/134.0.147398.0.html
Interview du français Vincent Paquet, co-fondateur de GrandCentral. Il y décrit le système qu'il propose. *29/09/2006*

Juridiques :
[13] http://www.oecd.org/dataoecd/18/12/37329847.pdf
Rapport de M. Jaebum LEE de la Direction de la science, de la technologie et de l'industrie de l'OCDE (Organisation de coopération et de développement économiques) sur les implications de la VoIP pour les politiques.

[14] http://www.itu.int/ITU-D/treg/VoIP-fr.pdf
Nouvelle de l'ITU (International Telecommunication Union) sur la réglementation de la VoIP.



SPITSink

Résumé exécutif

Suite à de nombreux travaux depuis son introduction il y a une dizaine d'années, la technologie VoIP qui constitue l'avenir de la téléphonie, notamment d'entreprise, est arrivée à maturité. Les besoins en termes de qualité de service ont été solutionnés et garantissent maintenant un fonctionnement adéquat aux exigences de la téléphonie d'entreprise. Des standards tels que SIP ont été développés et permettent l'interopérabilité et par conséquent l'implantation massive et facile d'équipements de ToIP. La corrélation de ces éléments a permis l'apparition de pôles de compétences et de recherche actifs et performants. Par la même occasion, les hackers développent jour après jour de nouvelles menaces face auxquelles il faut se prévenir. Hier les spams par courriers électroniques envahissaient et polluaient vos boîtes mails, demain ce seront des appels non sollicités, du télémarketing, des robots programmés pour diffuser des enregistrements publicitaires, qui inonderont les réseaux VoIP et feront offices de spams vocaux dont l'appellation exacte est SPIT ou bien pire encore seront utilisés à des fins d'ingénierie sociale de manière à récupérer certaines informations. Dans ce contexte à la fois de foisonnement technologique et de menaces avérées, constatées et pour l'instant non solutionnées, Zeneo, jeune entreprise de 6 ingénieurs INSA Lyon, choisit de créer SPITSink dans le but de garantir efficacité d'utilisation et confiance dans l'utilisation de la ToIP.

En se basant sur des techniques à la fois statiques telles que des listes d'autorisations ou bien dynamiques telles que le calcul probabiliste ou la surveillance en temps réel de l'activité sur le réseau, SPITSink garantira le blocage des appels indésirables. De plus il permettra à l'utilisateur de fournir des commentaires suite aux appels reçus afin d'affiner les règles de filtrage et ainsi définir une politique d'administration du réseau ToIP adaptée.

SPITSink se présentera sous la forme d'un logiciel qui s'installera sur l'IP-PBX concerné et s'intégrera en totale transparence avec celui-ci. Ainsi la qualité de service de la ToIP sera maintenue, les besoins des utilisateurs en termes d'utilisation sûre et sereine seront satisfaits et enfin la sécurité du réseau VoIP sera assurée. C'est en comprenant les technologies et les besoins de demain que Zeneo avance et propose des solutions sûres pour un monde plus sûr.

1

ZENEO

A.3 Fiche d'évaluation

Nom/Numéro du projet :

Cette fiche est anonyme. Elle est envoyé au chef de PI, et est retransmise aux tuteurs et au chef de projet.
****

Quelle note globale donnez vous au projet d'un point de vue technique ?

A - Je soutiendrai entièrement ce projet (Acceptation)
B - Je trouve le projet interessant (Faible Acceptation)
C - Je pense qu'il est un peu défendable (Faible Rejet)
D - Je ne vois pas l'interêt (Rejet)

Commentaires :

Quelle note mettez vous sur le planning proposé ?
A - Il tient bien la route
B - Certains points me semblent légers
C - Je pense qu'il y a quelques éléments à changer
D - Je ne vois pas comment ca peut entrer dans ce planning

Commentaires :

Quelle note mettez vous sur le placement économique du projet
A - C'est un positionnement irrefutable (Millions de clients)
B - Ca doit passer sans problème (100 aines de clients)
C - Certains peuvent être éventuellement interessés (10aines de clients)
D - No vraiment je ne vois pas

Commentaires :

Quels sont les informations supplémentaires que vous voules faire remonter à l'équipe (forme, questions de détails, précisions).

Quels sont les 2 points forts (ou plus)

Quels sont les 2 points faibles (ou plus)

Bon pour finir, une note générale sur 20 :

A.4 Poster de projet

1.5 Geek attitude

Je suis programmeur depuis l'âge de 14 ans. J'ai essayé la programmation dans de nombreux langages, BASIC, LISP, FORTRAN, PERL, SHELL, C, C++, JAVA, RUBY, CAML, PYTHON. J'essaye de m'auto-former sur un langage tous les ans. Je suis très intéressé par la théorie des langages et leur évolution même si ce n'est pas nécessairement le cœur de ma recherche.

J'ai également géré et largement étudié la majorité des systèmes d'exploitation modernes principalement Unix (AIX, HP/UX, SOLARIS, SUNOS, LINUX, NT, XP, UCLINUX). Je suis utilisateur de Linux depuis 1994 et j'ai utilisé un certain nombre de distributions depuis (SLACKWARE, REDHAT, DEBIAN et maintenant GENTOO). J'interviens dans un cours sur les systèmes d'exploitation pour parler de l'historique des systèmes, ainsi qu'en formation du premier cycle INSA pour motiver les étudiants aux réseaux et aux systèmes. J'ai une bibliothèque personnelle de 250 livres d'informatique et assimilés. Je suis abonné à Dr. Dobbs magazine depuis 15 ans, à Wired depuis 7 ans. Mes compétences techniques et ma flamme pour la transmission des compétences techniques sont parmi mes points forts.

Chapitre 2

Selection

Voici une sélection d'un article par projet.
- MUSE : [Royon et Frénot 2007]
- MOSGi : [Frénot et al. 2008]
- SOSGi : [Parrend et al. 2007]
- VOSGi : [Royon et al. 2006]
- DARTS : [Brebner et al. 2005]

2.1 Projet MUSE [Royon et Frénot 2007]

Yvan Royon and Stéphane Frénot

Multiservice Home Gateways : Business Model, Execution Environment, Management Infrastructure

In *IEEE Communications Magazine*, pages 122–128, 2007.

Multi-Service Home Gateways: Business Model, Execution Environment, Management Infrastructure

Yvan Royon and Stéphane Frénot, INRIA Ares / CITI, INSA-Lyon, F-69621, France

***Abstract*— The Home Gateway market is undergoing deep changes. On one side, home networks are evolving, getting dynamic and federating more and more devices. On the WAN side, new actors appear, such as multimedia content providers. From both sides emerge new features and new management needs.**

In this article, we highlight challenges and benefits of moving more intelligence to the Home Gateway, making it more than a simple interconnection device. We argue that we need a full-fledged execution environment on gateways, to address the evolution of business models. We present this evolution and summarize existing solutions for execution environments and management for Home Gateways. Then, we propose two improvements that reflect the new requirements. The first improvement is a high-level virtualization of service gateways, and the second one recommends an end-to-end dynamic multi-provider management system.

I. Introduction

During the last years, the Home Gateway market has evolved at a very fast pace. For a time, it only consisted in bringing IP connectivity to the home. Available services were common application-level programs, such as Web or e-mail clients. Today, operators are moving to integrate value-added services. Multicast TV and Voice over IP services are largely advertised; their integration with data transport is referred to as *triple play*. These three network-enabled services are provided either through the same connectivity box (the modem) or through a dedicated set-top box.

In the close future, operators plan to open the service delivery chain[1]. The roles currently assumed by the Internet Access Provider will be split between connectivity provisioning and service provisioning. Instead of being tied to a single service provider for television and phone over IP, the end-user will have a variety of choices and will gain from competition, both in price and diversity. This business model is referred to as *multi-play*.

A model with multiple actors outlines new challenges. The Home Gateway hosts a set of configurable services, which can be operated, controlled or monitored by different Service Providers. However, these important issues are not sufficient to get the full picture. Indeed, voice and video are not the only services that can be of interest for home users: domotics, entertainment, home security and health care are also potential markets. For a long time these other services have been considered individually, with separate networks, devices and user control. By integrating them with the home network, new use cases are possible.

As an example, a network-enabled white good (e.g., an oven or a dishwasher) can be monitored by its manufacturer. In case of failure, for instance if the temperature in a fridge rises above a certain threshold, an alarm would be raised to the home user and to the nearest repairman. The contents of a fridge or a wine cellar can also be monitored using sensors. The home user would input preferences, or orders would be sent to the nearest retailer. Many current projects focus on providing help for seniors or for the disabled at home; these issues are known as *quality of life*. For instance, the Home Gateway would monitor a pacemaker or similar medical appliances, and alarms would be forwarded to family members and the nearest hospital.

All these examples show that the definition of "service" must be broadened. It should not be limited to network multimedia flows, but also include management facilities for in-home devices (configuration, preferences, monitoring), human-machine interfaces, and any kind of application the Home Gateway may host. This paper refers to this extension as *multi-service*. We keep this definition open so that future ideas and applications can be seamlessly integrated.

The Home Gateway must undergo changes so that the multi-service model can be implemented. Indeed, home users should be able to use in-home devices and set preferences regardless of the status of the access provider's network. This means that most home-related settings should be hosted inside the home. In the triple play model, the Home Gateway already hosts or uses information and configuration from both home users (e.g., WiFi access point settings) and the Access Provider (e.g., last mile settings). Therefore, it seems a natural choice to enhance Home Gateways to be multi-service ready.

Our goal in this paper is to provide mechanisms for multi-service enhancements, which target both the OS layer and the management plane. Figure 1 shows a management perspective of the multi-service home environment. Three "realms" interact around home gateways. The Access realm is the usual Access Provider, who provides layer 1-4 connectivity to the gateway. The User realm contains all interactions the end-user and local devices may have with the Home Gateway. Thirdly, the Service Provider realm represents the interactions between a Service Provider and a service hosted on the gateway.

From an execution layer point of view, these multiple realms and entities also coexist on the gateway. This means that some level of software isolation must be defined.

[1]See the MUSE European Project, IST-FP6-507295, which partially supports these works. http://www.ist-muse.org

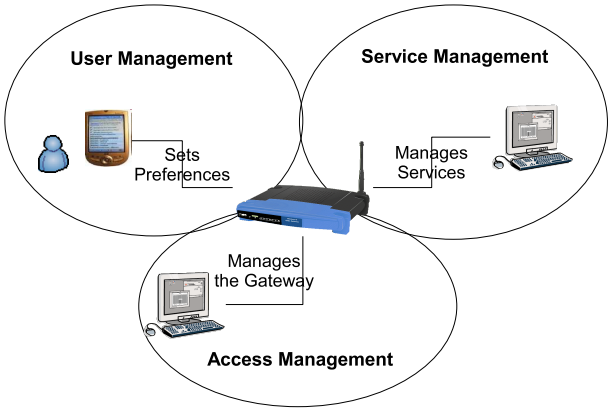


Fig. 1. Management Realms

The article is structured as follows. Each section focuses both on the management layer and the execution environment. Section II presents existing solutions related to Home Gateways. Section III defines new requirements that result from the multi-service model, and the enhancements that must be made compared to existing solutions. Section IV describes our testbed implementation of a multi-service gateway; results are detailed in section V. Finally, section VI concludes the article.

II. Overview of Existing Home Gateway Environments

New business models bring new technical requirements. With the connectivity-only model, an Auto-Configuration Server (ACS) provides layer 1-4 parameters to the Home Gateway (e.g., DHCP). There is only one ACS, hosted by the Access Provider. With the triple play model, the ACS must store additional intelligence and parameters related to voice and video subscriptions. Moreover, it manages not only the Home Gateway, but also other Customer Premises Equipments (CPE), such as set-top boxes for TV. With the multi-play model, there might be more than one ACS. Coherence and aggregation of management activities from different Service Providers to one particular Home Gateway become a problem.

With the multi-service model described above, it is clear that more intelligence must be put inside the Home Gateway, making it a pivotal device in the home network. In this section, we first summarize existing management solutions that evolve around the Home Gateway. Then, we present a panel of execution environments applicable to this particular device.

A. Management Solutions

With the triple play model, home users configure their gateway and their services through the Access Provider's Web site. However, with the multi-service model, there can be more than one service provider; the home user may also interact with in-home devices, whether the Internet access is active or not. As a result, a single management server, located outside of the home, is no longer desirable. The solutions that exist for this problem can be categorized in two groups: WAN management protocols and application / service management protocols.

WAN management protocols are network-centric solutions that follow a three-tier architecture: the manager (e.g., an ACS), the agent (on the managed device), and the connector (for translation between the former two). Existing management technologies answer three questions: how the data is structured, which data is available, and how the agent and the connector interact. Common examples ares TR-069 from the DSL Forum, CIM/WBEM from the IETF, and SNMP.

Application management protocols focus on home device management, such as service discovery and autonomous service description. Examples are JMX in the Java world and UPnP.

An implementation of the multi-service model can make use of several of these WAN and application management solutions. The other side of the story is that the Home Gateway must support the management agent, but also other applications, configuration facilities, etc. Such support is provided by the execution environment.

B. Execution Environments for Home Gateway

With triple play and multi-play business models, Home Gateways are becoming more than just modems: they are resource-constrained computers. Their task is not only to commute network packets, but also to filter them, to support multicast, to be upgradeable, to host application services and to support user interaction. Two kinds of execution environments are popular in academia and industry: GNU/Linux and OSGi.

1) Linux: The obvious trend at the moment is that constructors are abandoning proprietary operating systems that used to power their modems, in favour of Linux variants. There are several strong advantages.

- Linux exists in different flavours, such as uCLinux, which makes it adaptable to new architectures [1], including resource-constrained environments;
- Source code is open and mainstream, and so is tested by lots of users;
- The programming model is standard C or C++. Finding able developers is very easy;
- It is then trivial to reuse existing applications for networking (firewalls), multimedia (codecs), or management (agents for various protocols, such as SNMP).

Current operating systems still suffer drawbacks in multi-service environments. Integrating applications from different providers, making them work together and ensuring their correct behaviour is time consuming. One possible answer is to use type-safe languages and sandboxing techniques. The mobile phone industry went this way with Java. The Home Gateway industry is following the same path with OSGi.

2) OSGi Technology: In its early stages, OSGi was designed specifically for open service gateways[2]. The OSGi service platform is a container built on top of a Java virtual machine. It hosts deployment units called *bundles*, which contain code, other resources (pictures, files), and a specific start method. A descriptor file expresses dependencies on other software pieces, among other meta-data.

[2]http://www.osgi.org

The OSGi platform automatically checks and resolves dependencies among bundles. It also dynamically controls bundles' life cycle, and enables hot deployment and update. This simplifies the administrator's work and improves stability.

The programming model is Service-Oriented Programming (SOP) [2], which derives from Object-Oriented Programming (OOP). The driving motivation is to minimize coupling among pieces of software. A service is an interface, or a contract, that describes what a piece of software does. How it is implemented is hidden internally.

The OSGi specifications keep the management layer open, in the sense that it does not impose a specific management protocol or technology. In real-life situations, several technologies can be combined to provide an end-to-end management solution [3].

OSGi is getting popular in academia communities, but more so in industry. For instance, some BMW cars use OSGi for non-critical tools. Another example is the Eclipse development kit, which rebuilt its whole plug-in architecture around an OSGi framework, and most J2EE implementations, which are or plan to be redesigned as OSGi bundles. Actors interested in Home Gateways are also adopting OSGi, through the Home Gateway Initiative (HGI), a telco-driven consortium.

3) HGI: The Home Gateway performs various tasks, from protocol translation to QoS enforcement to remote access. Since most of these tasks are dynamic, configurable, manageable, and can change over time, they should be implemented by updateable software. HGI [4] has defined Operating System requirements for Home Gateways, which are largely based on the OSGi specifications. They operate on three levels.

The first one is software management. It includes:

- Configuration of software pieces, called "modules";
- Management of their life cycle: install, update, uninstall;
- Dynamicity and security enforcement. Modules are registered and verified; their dependencies are resolved, then linked;
- Resets for default configuration parameters, and for firmware debug in case of low-level failures.

The second level is performance management and diagnostics, which include:

- Remote diagnostic tests for hardware and software elements;
- Performance monitoring;
- Support for events sent by the gateway.

The third and last level gathers definitions of users in presence:

- A super-user (the Auto-Configuration Server), in control of everything that is manageable;
- A local administrator, in control of local management (e.g., firewall, users);
- End-users, with permissions set by the local administrator.

Current works on Home Gateways are led by the HGI specifications, the OSGi platform and management technologies. And yet, they lack the integration of the multi-service business model. They still focus on models with a single Access and Service Provider, hosting a single ACS. Our goal is to enable the multi-service model, from both the management layer and the execution environment points of view. The next section lists the requirements for these two issues.

III. Requirements for Multi-Service Home Gateways

A multi-service Home Gateway hosts various services, applications, etc. from several Service Providers. The fact that these Service Providers can be separate entities impacts both the management layer and the execution environment.

A. Management Requirements

Figure 1 shows that the actors around the Home Gateway use the management layer for different purposes. In the multi-service model, management should not be classified according to network span (LAN, WAN) or OSI layer (IP, TCP), but according to the goals and features of each actor.

For example, a home user will set preferences and configure devices using UPnP technologies, Flash interfaces, or proprietary solutions. An Access Provider will configure layer 1-4 parameters and manage QoS using TR-069. Service Providers may prefer JMX or SNMP to monitor their services.

With these examples in mind, we can express management requirements as follows:

- Each actor around the gateway (home user, Access Provider and each Service Provider) can use his own dedicated management agent;
- Technologies used for management agents are heterogeneous.

These two points allow to isolate management activities from different actors, in terms of network flows. We must also address a separation local to the gateway, in terms of runtime execution.

B. OS Requirements

HGI requirements work well in a mono-provider scenario, with a single ACS. However, they fall short on several points in the case of an open, multi-tier environment. Indeed, different Service Providers, potentially competitors, may deploy modules on the same Home Gateway. These modules may contain code or data that are business-critical, or simply that give away information on the client base, configurations, or implementation performance. They may also contain malicious or buggy code, which could endanger all modules in presence.

Therefore, the need for isolation among actors present at software level dictates the following additional requirements.

- Definition of users
  - Service or Software Providers can be seen as users on the Home Gateway, similarly to multi-user execution environments. These users need a secure authentication phase and a secure session for all activities on the gateway, such as module deployment;
  - A Root user controls users allowed on the system. He also checks the overall resource consumption, and may take coercive measures against users that threaten to starve resources.

- Isolation and sharing of modules
  - Modules from different providers (users) must be isolated by default, i.e., a user can only see and interact with his own modules;
  - A single instance of some modules may be shared among all users on the Home Gateway. This is useful for libraries (such as codecs), or for common modules (such as a Web server).
- Remote access, for management features.
- Preferences and configuration can be stored locally, in the case that they represent private information or are not related to an ACS.

In this section we described the changes that multi-service gateways must undergo. The next section shows how we implemented these changes using OSGi.

IV. A Framework Implementation for Multi-Service Home Gateways

OSGi offers important features for Home Gateways: service-oriented programming, dynamic life cycle management, dependency checking. Unfortunately, it lacks concepts for the multi-service business model. To enable this model, we propose improvements on two levels. At the run-time level we implement a virtualization of the OSGi framework (see [5]); at the management level we provide an end-to-end, JMX-based solution.

A. OSGi Virtualization

Software elements that belong to different providers must be separated. In other words, we need some level of runtime isolation. Various works already exist on this topic, such as BSD jails, Xen, VServer or VMware. They propose different levels of enforcement in terms of resource isolation and namespace isolation, which are a trade-off with performance.

For instance, Xen [6] runs separate copies of a whole operating system for each isolated environment. VServer [7] does not duplicate the kernel, but still duplicates inodes on the filesystem for each isolated environment. A BSD jail will duplicate only a portion of the filesystem, but will not provide a strong resource isolation.

In our case, following the multi-service model, the chosen level of isolation must still allow to share services on demand among different Service Providers. This means that isolation should be permissive. Moreover, we focus our implementation on Java/OSGi environments. As a consequence, sharing applies to OSGi services (which are Java interfaces). For scalability reasons, duplicating a whole JVM for each isolated environment seems overkill. Lastly, a Home Gateway has limited resources; typically 16 MB of permanent storage and 64 MB of RAM.

These reasons led us to opt for a weaker but lighter level of isolation than Xen's or VServer's.

Our approach is to embed OSGi as an OSGi bundle. A "core" OSGi instance runs multiple OSGi instances, all sharing the same JVM. Then, the scalability factor is only the price of virtualizing (or embedding) OSGi.

The aim here is to give each Service Provider a separate execution environment, as depicted in figure 2. A virtual gateway is operated by one Service Provider, who sees it as a standard OSGi service platform.

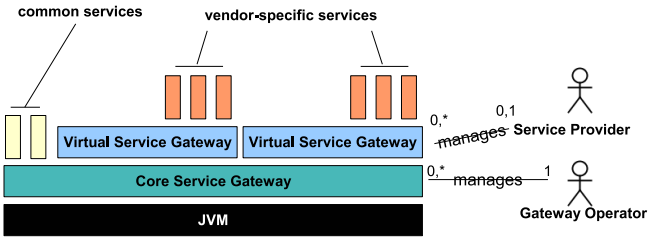


Fig. 2. Multi-Service, Multi-Provider Home Gateway

The core service gateway is responsible for launching virtual gateways. It is managed by a gateway operator, who can also be the owner of the gateway or the Access Provider. Operations related to virtual gateway life cycle are subject to contracts between a Service Provider and a gateway operator.

We now have an execution environment where each actor around the Home Gateway gets a little privacy. Service Providers are only aware of services running in their own virtual gateway. The gateway operator ensures that allowed virtual gateways are up and running, but cannot access their contents.

After ensuring isolation, we need to add a controlled way to share code and services. Two cases are considered. The first one is services common to all actors on the Home Gateway. This could be useful for an embedded Web server or a generic fault logger. These common services are hosted by the core gateway, and explicitly exported to all virtual gateways. When a virtual gateway is launched, it adds this collection of shared services to its internal service registry. The second scenario is for sharing code, such as libraries. For example, a single instance of multimedia codecs can be hosted by the core gateway, and shared with all virtual gateways. It is similar to the previous case, except for the import/export mechanism in OSGi (Java package versus OSGi service).

So far, we have described an execution environment that supports the multi-service model. The following describes the management framework that runs on top of it.

B. An End-to-End JMX-Based Management System

Multi-party access to a gateway can be handled using different approaches. One is to create a specific entity naming scheme for each provider. That means that each provider may access only a subset of the whole management data. We chose to define a virtual execution environment for each Service Provider; thus, a Service Provider sees a gateway as if he was the only one using it. Each virtual gateway hosts its own management agent, with the technology the Service Provider chooses.

Our default implementation uses JMX. For scalability reasons, each JMX agent needs to be lightweight in terms of

memory consumption. We use a modified version of MX4J that we stripped down and split using Service-Oriented Programming.

Management data in JMX is represented through MBeans, which are Java interfaces. They follow the JavaBeans model: get an attribute, set an attribute, execute a method. Queries on an MBean are redirected to a set of probes; we have implemented probes that give access to:

- The OSGi framework: version number, current running profile, etc;
- Bundles life-cycle: start, stop, update and list bundles inside the current gateway;
- OBR, the Bundle Repository: allows to trigger the installation of a bundle on a remote gateway;
- Memory usage inside the Java virtual machine;
- The underlying OS metrics: global CPU, memory, swap and disk usage.

Core gateways also have an MBean to start, stop and list running virtual gateways.

Any bundle can come with its own MBean, for instance to represent management data for a specific device or OSGi application. In this case, the MBean registers itself to the OSGi framework as an OSGi service. The JMX agent, listening to service-related activities, automatically registers the MBean; this effortless approach is called the *white board* pattern [8].

Finally, a remote logger implements the OSGi Log service. It notifies a remote manager of each event that occurs on the gateway.

The JMX agent and all MBeans are located on the gateway. For remote interactions, queries on MBeans are sent through connectors. In our case, RMI and XML/HTTP are used.

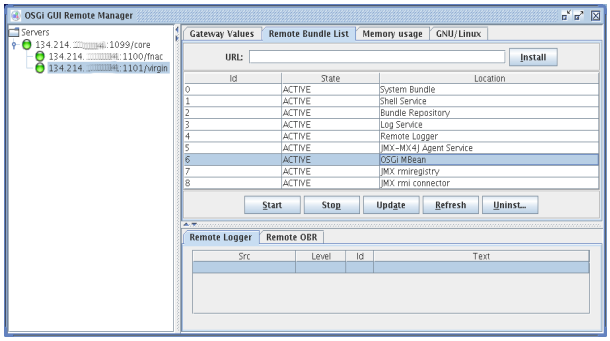


Fig. 3. Monitoring Console

We have developed a monitoring console (figure 3) that dynamically discovers the list of bundles on a gateway. Their associated MBeans are gathered following the *visitor* design pattern [9], then displayed in graphical tabs. This allows to specialize management depending on the user's service subscriptions, and to reuse the console for each of the three realms presented in figure 1.

V. Results

A. Memory Performance

Figure 4 shows the amount of memory used on our test system[3], cumulating OS, Java, OSGi and applications . On the x axis, we run sample services that each allocate 1 MB of memory[4]. On the first curve, all services run in the same gateway (single-user mode). The other curves run each service in a separate virtual gateway, with and without separate management agents.

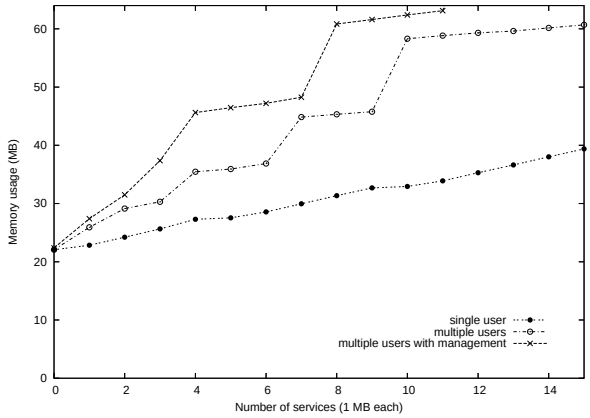


Fig. 4. Memory Performance on a 64 MB machine

Our current implementation, which is not yet optimized for size, shows a 3 MB memory overhead per service provider using OSGi virtualization and a JMX agent. There are no significant CPU and disk overheads.

The memory overhead is not negligible, so there is still room for improvement. However, the advantage of this proposal is to enable OSGi's service-oriented programming between multiple virtual gateways, without modifying the underlying OS and JVM. Other options suffer from either scalability, programming model or availability. The first alternative is to launch one JVM per user; scalability is one order of magnitude worse than OSGi virtualization, and the OSGi framework would need heavy modifications to support SOP between JVMs. The second alternative is to use a multi-task JVM. This is the best compromise between scalability and resource isolation; however, the only implementation that we know of [10] runs only on big SPARC/Solaris systems. A last alternative is to use lower-level isolation such as Xen. This offers the best resource isolation available; however, service-oriented programming between Xen instances is yet to be achieved. With current implementations, we would need to run one JVM per Xen instance, with the downsides cited above.

[3]An average Home Gateway has a processor around 266 MHz, 64 MB of memory and 16 MB of storage. We have run performance tests on an Epia (1 GHz Nehemiah CPU, 64 MB of RAM) running a Sun JDK 1.5 with default parameters, with the Felix OSGi implementation.

[4]1 MB is enough to host e.g., several small games, embedded applications and tiny web servers. Our own UPnP/Audio-Video control point uses 600 KB of memory.

B. Management Performance

The management layer is evaluated by measuring response times. On figure 5, the thin black curve is the pseudo-theoretical load induced by a simple probe : `getCPU()`. Its shape is $y = \alpha/x + \beta$. α can be seen at $y = 100\%$ CPU; it is the response time for `getCPU()` with zero network delay. β is the residual load when the gateway is not running any particular application; this is also called system noise. The curve is obtained by measuring two arbitrary points, e.g., at 100 and 400 ms period.

The thick green curve is the experimental load. It calls the `getCPU()` probe at various periods, ranging from every 1000 ms to every 10 ms. The peaks on the curve correspond to garbage collections on the system under scrutiny.

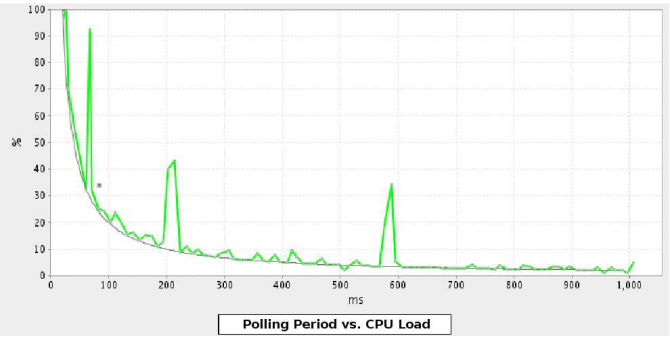


Fig. 5. Load Induced by Management Probes on a 133 MHz machine

The figure shows that, if we allow the management system to use no more than 5% CPU time, the `getCPU()` probe can be polled every 500 ms. With around 10 users on the gateway, each Service Provider can poll a simple probe every 5 seconds without encumbering the CPU.

C. Summary

Looking back at the requirements expressed in section III, we have addressed some of them, and left others open. Here is the status of what works and what needs improvement.

1) Technical definition of users: Users (i.e., service providers) are defined as managers of virtual gateways. Each user has his own management agent and management data, in his own virtual gateway, accessible via his own port number. Thus, users are isolated from a management point of view. Moreover, each user chooses his own technology; we use JMX, but SNMP bundles are available on the OSGi Bundle Repository[5].

We have not yet addressed user authentication and secure session. We plan to use SSL-enhanced connectors for this.

The last requirement in this category was that the root user is able to take coercive measures against users that use too much resources. This needs further works, since resource control for Java-based embedded systems is still a hot topic. We plan to use a feature similar to Linux's OOM-killer (Out-Of-Memory): when too much memory (or CPU) is used system-wide, root locates the most consuming thread and kill its virtual gateway. If the thread belongs to root, the related bundle should be killed.

2) Isolation and sharing of modules: We enforce namespace isolation between software modules (OSGi bundles) that belong to different Service Providers. This is weaker than resource isolation, but has the advantage of being OS- and JVM-independent, and it is production-ready.

Service sharing has been implemented statically. When the root user creates a new virtual gateway, the core gateway declares a list of services to export. In the future, we plan to export this list dynamically.

3) Remote access for management: Our management bundles contain connectors for RMI and HTTP. Connectors for other remoting protocols are possible.

4) Local storage for preferences and configuration: We use a straightforward scheme to isolate local storage among users. The Felix OSGi implementation uses profiles to cache bundles and data related to bundles. Each virtual gateway has its own profile, with its own cache directory. Bundles usually access files via the `bundleContext.getDataFile()` OSGi primitive, which only allows to reach this private directory. We still need to stop malicious bundles that try to open `FileInputStream`s on other users' cache; Java permissions allow to do this.

Our code is available under open source licenses. Management-related projects are integrated within the Apache Felix project [11], distributed under the ASL license. The code for OSGi virtualization is also written for Felix, and can be obtained on demand under the CeCILL license.

VI. Conclusions

Business models around Home Gateways are evolving, and will inevitably open the door to new actors and new services. The immediate conclusion is that the execution environment on the gateway must be split into isolated areas dedicated to different Service Providers, and that each actor needs autonomy and choice in terms of management solutions.

We propose a lightweight, OSGi-level isolation of the execution environment, where we launch "virtual" gateways (one per Service Provider) inside a "core" gateway (controlled by the operator). Each core and virtual gateway runs a separate management agent. This enables namespace isolation; it is weaker than Xen's or VServer's isolation mechanism, but it scales reasonably on resource-constrained devices and allows to share OSGi services between core and virtual gateways. On top of this, we provide an end-to-end management infrastructure (probes, agent and monitoring console) that take virtual gateways and multi-service constraints into account.

We show that a simple implementation allows to host in the order of ten Service Providers on a typical home gateway, with enough memory to run e.g. a UPnP/Audio-Video control point. Each Service Provider can query simple probes in the order of 10 times per minute without hindering the CPU.

[5]http://www2.osgi.org/Repository/

References

[1] Nicolas Fournel, Antoine Fraboulet and Paul Feautrier, "Booting and Porting Linux and uClinux on a new platform," Research Report 206-08, LIP / ENS Lyon, February 2006.

[2] Guy Bieber and Jeff Carpenter, “Introduction to Service Oriented Programming,” 2001. [Online]. Available: http://openwings.org/download/specs/ServiceOrientedIntroduction.pdf
[3] Juan C. Dueñas, José L. Ruiz and Manuel Santillán, “An End-to-End Service Provisioning Scenario for the Residential Environment,” IEEE Communications Magazine, vol. 43, no. 9, pp. 94–100, September 2005.
[4] Home Gateway Initiative, ”Public Deliverable v1.0,” July 2006. [Online]. Available: http://www.homegatewayinitiative.org/publis/HGI_V1.0.pdf
[5] Yvan Royon, Stéphane Frénot and Frédéric Le Mouël, “Virtualization of Service Gateways in Multi-provider Environments,” in Proceedings of the 9th International SIGSOFT Symposium on Component-Based Software Engineering (CBSE 2006), June 2006. LNCS 4063, pp. 385–392.
[6] Paul Barham, Boris Dragovic, Keir Fraser, Steve Hand, Tim Harris, Alex Ho, Rolf Neugebauer, Ian Pratt and Andrew Warfield, “Xen and the Art of Virtualization,” in Proceedings of the ACM Symposium on Operating Systems Principles (SOSP 2003), October 2003.
[7] Stephen Soltesz, Herbert Pötzl, Marc E. Fiuczynski, Andy Bavier and Larry Peterson, “Container-based Operating System Virtualization: A Scalable, High-performance Alternative to Hypervisors,” in Proceedings of the EuroSys Conference, March 2007.
[8] OSGi Alliance, “Listener Pattern Considered Harmful: The Whiteboard Pattern,” 2nd revision, 2004. [Online]. Available: www.osgi.org/documents/osgi_technology/whiteboard.pdf
[9] E. Gamma, R. Helm, R. Johnson and J. Vlissides, “Design Patterns, Elements of Reusable Object-Oriented Software,” Addison Wesley, 1995.
[10] Grzegorz Czajkowski, Laurent Daynès and Ben Titzer, “A Multi-User Virtual Machine,” in Proceedings of the USENIX 2003 Annual Technical Conference, pp. 85–98.
[11] Apache Software Foundation, “Felix OSGi R4 Service Platform implementation.” [Online]. Available: http://felix.apache.org/

2.2 Project MOSGi[Frénot et al. 2008]

Stéphane Frénot, Yvan Royon, Pierre Parrend and Denis Beras

Monitoring Scheduling for Home Gateways

In IEEE/IFIP Network Operations and Management Symposium (NOMS 2008), Salvador Bahia, Brazil, April 2008

Monitoring Scheduling for Home Gateways

Stéphane Frénot, Yvan Royon, Pierre Parrend and Denis Beras
INRIA ARES / CITI, INSA-Lyon, F-69621, France
tel. +334 72 43 64 22 - fax. +334 72 43 62 27
{firstname.lastname}@insa-lyon.fr

***Abstract*— In simple and monolithic systems such as our current home gateways, monitoring is often overlooked: the home user can only reboot the gateway when there is a problem. In next-generation home gateways, more services will be available (pay-per-view TV, games...) and different actors will provide them. When one service fails, it will be impossible to reboot the gateway without disturbing the other services.**
We propose a management framework that monitors remote gateways. The framework tests response times for various management activities on the gateway, and provides reference time/performance ratios. The values can be used to establish a management schedule that balances the rate at which queries can be performed with the resulting load that the query will induce locally on the gateway. This allows the manager to tune the ratio between the reactivity of monitoring and its intrusiveness on performance.

I. Introduction

During the last years, the Home Gateway market has evolved at a very fast pace. The business model has evolved from plain IP connectivity to triple play (voice, video and data). These 3 services are provided by a single entity: the internet access provider. In the close future, the business model will move to multi-play [1], and enable the management of smart homes [2]. The idea is that different sets of services, such as TV on demand, games, or home security, can be developed and deployed by various business entities other than the access provider. To push this idea further, different services will be delivered by various providers depending on the home user's choices.

Such a business model has a strong impact on their underlying technical infrastructures. Indeed, each service provider should be able to manage his own services. For instance, if the service is to monitor a pacemaker for an elder person, the hospital (or whoever is providing the service) should access the pacemaker's data as directly as possible. It is not acceptable that monitoring data is lost because the grandchildren are watching TV.

From the network point of view, the solution is to grant a certain Quality of Service to each provider, or even to each service. However, we must also look at the system point of view: home gateways have limited processing power, and management activities do have an impact on CPU utilization. Simply put, the more often a manager sends requests, the more accurate management data he will get. However, the CPU load will increase, until a threshold where services will not be able to run correctly.

We propose a management framework for home gateways that determines this threshold. It is implemented on top of Java/OSGi, using JMX. Section III gives a quick overview of JMX and OSGi. In section IV, we show how we can evaluate the cost of a `getAttribute()` request, namely a request to get the CPU usage on a gateway. Finally, section V describes a way to schedule management activities so that their cost in processing power are under control. Section VI concludes this work. [1]

II. RelatedWorks

We now present the state of the art in management of Java-based systems. An overview of the existing solutions is provided, and the JMX management opportunities, that emerges as one versatile solution for java applications. Until recently, the principal management technology has been the SNMP protocol. It brings with it a complete management framework, in particular data handling facilities with the MIB (Management Information Base). SNMP can also be used in home systems [3], as JMX is. However, no real integration with applications is provided, which makes the latter approach more relevant when high-level software systems must be managed. However, application management must be performed carefully, so as not to impair the system functions with management-related performance overhead. This question has been studied in the context of Web Services [4], and for EJBs [5], or with a focus on Operarting System management [6]. The performance question for OSGi management will be discussed in detail in this paper. JMX is the Java Management Extension [7]. It is meant for managing and monitoring Java-based systems, through a so-called Agent that supervises probes. The probes are accessed through MBean interfaces. JMX is part of the Sun Java project, and is the subject of two complementary specifications: the SUN JSR 3, 'JavaTM Management Extensions (JMXTM) Specification' 2, and the 'SUN JSR 160: JavaTM Management Extensions (JMX) Remote API' 3. Several tools have been developed to exploit the JMX functionalities. The JConsole [8] is the sun monitoring and management tool. MX4J 4, which is an Open Source implementation of JMX, also provides a set of JMX-related tools. So as to provide a full control over the managed systems, which is often built out of several elements, JMX must be integrated in a suitable framework. Jasmine5 is such a management framework for enterprise applications, developed in the frame of the ObjectWeb Consortium. It aims at managing the various parts of N-Tiers

[1]This work is partially supported by the IST-6thFP-507295 MUSE European Project

systems ,such as Java EE, Message orientedMiddleware, and Services Oriented Architectures). The intensive use of JMX for management brings scalability questions. This problem has been studied by [9], and will be further discussed in this paper. The specific instrumentation of OSGi platforms with JMX management facilities is more recent, and few powerful solutions exist. The Jasmine Project provides its own OSGi Console, which is rather simple, since it only lists OSGi Gateways and installed bundles. We developed a complete JMX Management Framework for the OSGi platform. The principle of our approach is presented in OSGi [10], and the core functionalities are shown is [11]. The current work introduces additional functionalities, and discuses performance optimization of JMX-based Management.

III. A Management Architecture for Home Gateways

In previous works, we have designed a JMX-based framework for home gateways [12]. It comprises a remote console, and software that runs locally on each gateway. In JMX, such software is:

- **JMX agent**: a singleton application that registers Java management interfaces called MBeans;
- **MBeans**: Java objects registered whithin the agent, and accessible by a public Java interface. Standard MBeans can provide 3 kinds of methods: get the value of attribute, set it to a new value, and invoke a method;
- **Connectors**: handled by the agent, they allow to access MBeans remotely, in our case using HTTP or RMI;
- **Probes**: feed management data to MBeans when performing get or set queries. Probes are implementation-specific, and are hidden by the MBean interface.

The role of the MBean is to provide information from the managed system to the remote manager. The information can be provided in two ways: through a solicitation from the manager (Response/Request), or through a notification from the MBean itself (Notification). The JMX specification enables various kinds of MBean (standard, dynamic, models and open) and various kinds of MBean behaviors (thresholds, relations, measurements); this study focuses on standard MBeans with get / set / invoke methods.

We have implemented these software parts as OSGi plugins (called bundles). The OSGi [13], [14] specifications define a framework that manages the life cycle of applications. These applications can be downloaded from a remote location, started, stopped, updated, removed. Finally, applications can also express dependencies towards other applications. The framework automatically checks that dependencies are met, and refuses to launch an application otherwise. The OSGi specifications do not specify any management architecture to control gateways. We have developed a JMX framework that enables the remote monitoring of OSGi based home gateways. The framework uses a JMX agent located on the framework, and an RMI (or HTTP) connector that enables the connection between an agent and a manager. We have also developed our own management console that connects to the agent. The entire framework is available in Apache Felix [15], an open source OSGi implementation. The management subproject is called MOSGi.

When managing a remote gateway, two interaction schemes are available: request/reply and notifications. With request/reply interactions, the manager periodically polls the gateway for values (memory, bandwidth...). With notfications, the gateway sends messages either on a timer basis or on a threshold basis. In both interaction schemes, management activities do have a cost in terms of processing power. This load corresponds to the various probes running on different threads and to the agent that maintains remote connections and a registry of current MBeans. Our concern is that the remote gateway has limited resources. These resources are shared between both user services (TV, voice, games) and management activities. This paper tries to elaborate a way to determine the volume of management activities that can be imposed on a home gateway without disrupting user services.

IV. Management Cost Evaluation

Managing home gateways has to cope with 2 opposite visions. When something goes wrong the gateway operator should be warned as early as possible. But in order to achieve this, the gateway needs to handle an extra management activity that burdens the gateway. The question is simple: supposing that a figure under 5% of resource consumption is not intrusive, how often can a manager query management data? The quantity is expressed as the number of requests per second (or minutes) that can be sent. This measure reflects the response time an administrator can expect on a remote device.

A. CPU measurement

As a first approach we are working in request/response mode. Figure 1 shows the CPU consumption that is induced when querying the CPU utilization ratio.

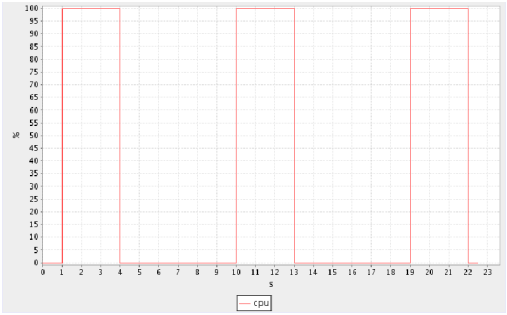


Fig. 1. Cpu Monitoring

In this example the polling is made every 9 seconds and the probe returns its value in 3 seconds. If the gateway does nothing else than reacting to this query, it should present a load of 33%. It is easy to understand that the load average is directly related to the execution time (in CPU cycles) of the `getCpu` method. In the Unix world, the `/proc/stat` file holds data needed by the getCPU probe. The generic pseudo-algorithm of the probe is the following:

```
public int getCPU(){
  open(/proc/stat);
  int currentIdleValue=readIdleValue();
  int cpu=(currentIdleValue-oldIdleValue)/Duration;
  oldIdleValue=currentIdleValue;
  close(/proc/stat);
  return cpu;
}
```

Listing 1. Pseudo-code for the getCPU probe

B. Determining the cost of the CPU probe

In order to manage a remote device, we need to know at which rate we can request data from it, and which quantity of data we can get in each request. The more requests by minutes, the fastest we can identify a fault, but the more load we put on the remote device. The theoretical curve obtained is shown in figure 2 (smooth curve). This is a direct way of obtaining the request rate the manager can apply. According to these estimations, if the manager asks for the cpu every 250ms he will generate a cpu load of 17%. If he asks every 2s the rate drops down to 2%. Of course if the observation is made every 2s, the system cannot identify the failure and the end-user can feel a service disruption.

The next section is a proposal to automatically obtain this curve. Knowing this curve, a manager can elaborate a management schedule among the various equipment available. This management planning should enable a fast decision making without disrupting the user's services.

V. A FRAMEWORK FOR MANAGEMENT SCHEDULING

A. Determining the cost of a management probe

As was presented in the previous section, we want to find a way to establish the consumption curve. The curve establishes the CPU rate involved by the management system. We have developed two approaches: the first one is empirical, and tries various requests rates; the second one is theoretical and finds significant points that allow to deduce the curve.

In the empirical approach, we define a series of rates at which the remote equipment is solicited. After setting these rates, the management console triggers a value request for each of them. This test, run on a LinkSys NLSU2 equipment, gives the second curve (not smooth) on figure 2. The values start at 97% which means that whatever the manager does, he will not be able to load the remote system more than 97%. We also see that there is an asymptote at about 0.5% which represents the average load without any activity. This load is achieved for a probing period of approximately 2s. So if we want not to disrupt the load of the managed system, we can make one request every 2s. A more important information is that we can see that the curve has the $\frac{\alpha}{x} + \beta$ form. Below we try to find the theoretical curve in a faster way, in order to determine the probing period that match 5% of system load (which can be considered as negligible) efficiently.

$\frac{\alpha}{x} + \beta$ is a trivial curve. The β parameter represents the load without activity and the α parameter represents the peek load that the management system can put on the remote equipment. For instance, if the period is 1s for a 100% load, it is trivial that for a 2s period the load should be 1/2 which correspond to 50% load (with the β parameter approximation). So the problem is to find a way to extract the α and β parameters of this curve. We have build an application that makes two measures and extrapolates the curve. For the first point, the application sends requests as fast as possible and for the second point, the application tries a long period of requests.

Figure 2 compares the two curves previously presented.

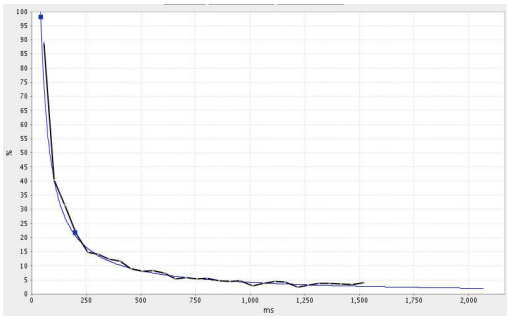


Fig. 2. Estimated load curve and empirical results

The smooth curve is the one deduced from the significant points. The other one represents the empirical measurements. We see that testing all values or taking only two measures lead to similar results. The next step is to choose the 2 points that allow to quickly calculate the estimated curve.

B. A management tool for evaluating the cost of a probe

We have integrated the tool to make these evaluation in our management architecture. It relies on a management agent residing on the remote equipment which reacts to requests from the management console. The management console has the following user interfaces:

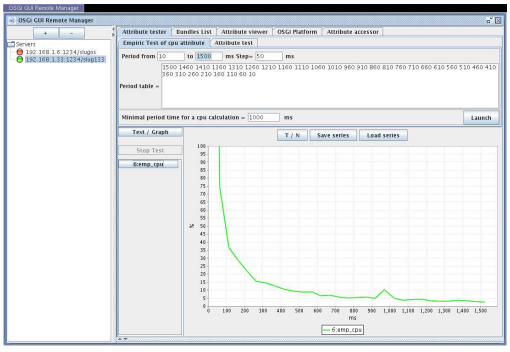


Fig. 3. Management Console: Empirical tab

The left part of the user interface (figure 3) enables the selection of the remote equipment. The right top part of the interface enables the input of parameters. In the sample the user wants to test request periods from 1500ms to 10ms of delay with a step of 50ms. When he validates the value, a series of testing periods is provided (the top Java TextArea). The minimal probing period is an important value, it tantamounts

to the maximal rate of information retrieval by the manager. If periods are too short, tests are biased with the time the probe needs to perform the calculation. One bias is linked to the access to `/proc/stat`: the delay can interfere with measurements. The other bias is linked to management timers: Linux cannot provide precise data if the querying is too fast. In order to absorb these two bias we define the Minimal period time for a CPU calculation value, which force the request/reply cycles at least to this value. The value corresponds to the minimum delay at which the system will be loaded for a period of time. Finally if the user clicks on the 'Launch empiric' button it will generate a series of queries for each period, and provide the corresponding curve.

The second tab "CPU attribute test" of this user interface drives to the $\frac{\alpha}{x} + \beta$ curve production. Figure 4 shows this interface. The upper zone enables the user to choose the tested

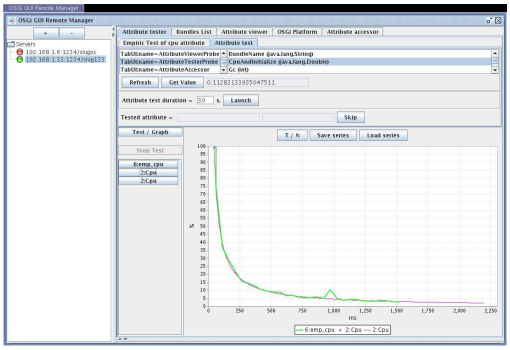


Fig. 4. Management Console: $\frac{\alpha}{x} + \beta$ tab

probe and the time duration of the test. When he presses the *launch* button, the system chooses automatically two periods to test and determines the α and β parameters from a curve extrapolation.

We implement some heuristics in order to choose these two periods:

- The second result should be at least 70% lower than the first one. If the first value generates 97% of CPU load, the second point should be evaluated somewhere where the result is lower than 29%.
- The same number of requests must be executed for the points determination. This balances the garbage collect activity.
- The system should control that the activity without management is stable during the tests. This means that before, between and after the tests, the activity is stable.

As a conclusion, our approach enables a fast evaluation of the management cost of the CPU probe. The curve can be determined with two significant points. In the next section, we extend this to the evaluation of all available probes.

C. A generalisation of scheduling for any management probe

In the previous section we showed that we could evaluate the cost of the CPU probe for a low end system. Since the remote equipment is managed in a multi-service environment we think that not only the CPU cost should be evaluated. In many business scenario the CPU load is not relevant and many other values can be more important. Significant values can be either "classical" ones, like available memory or bandwidth, but it can also be more specific ones such as "how many services are deployed" (platform management data) or "how many beers are in the fridge" (application level data). Provided we have the corresponding MBean , the framework is are able to evaluate the cost for any probe. We automate the process of evaluating any remote probe. Figure 4 shows the associated user interface. The service manager selects the probes he wants

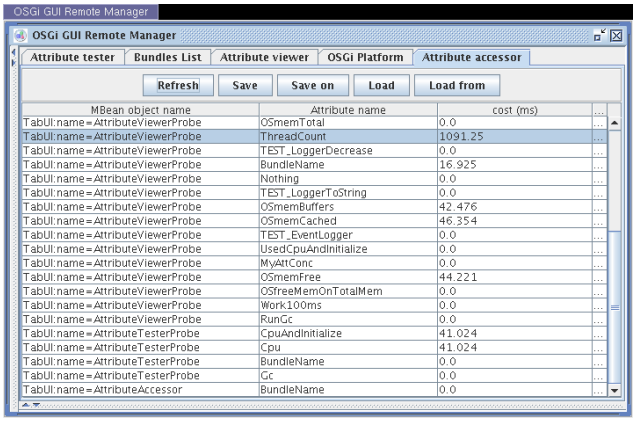


Fig. 5. Attribute CostResult

to evaluate. For each probe the framework evaluates the delay (minimal time between each probe request) that corresponds to an increase of 100% of the remote equipment load.

Figure 5 shows results for some tested probes. For example the OSmemFree probe costs 44.221ms which means that requesting OSmemFree probe every 44ms will increase gateway CPU activity by 100%. So If we want a 5% load, the corresponding period is *i.e.* 884ms. The 0.0 values just indicates that probes are not tested yet.

These data can be used to elaborate a management plan. This plan indicates which and when each management probe can be requested. For instance if the manager asks the OSmemFree value every 884ms and the Cpu probe value every 820ms we will have an increased CPU load of 10%. This management plan is a fundamental concept where the manager needs to find the right balance between the reaction time he wants and the load put on the remote equipment. The more reactive he wants to be, the more load he puts on the remote site. Establishing the plan a-priori can lead to a better anticipation of the system evolution.

D. Results

We validate our framework on three classes of systems: 1) an Intel dual core running a desktop environment, 2) an EPIA 1000Mhz simulating a high-end multimedia home gateway, and 3) a LinkSys NLSU2 representing a lightweight home gateway.

All these system runs Gentoo Linux with the JamVM virtual machine and Felix/MOSGi as the management system[2]. The first question to answer is : "is the theoretical curve compliant with empirical tests". Figures 6, 7 and 8 represent these two curves for the 3 test environments. The curve with square points represent the "empirical" approach the curve with round points is the curve deduced from α and β parameter measures.

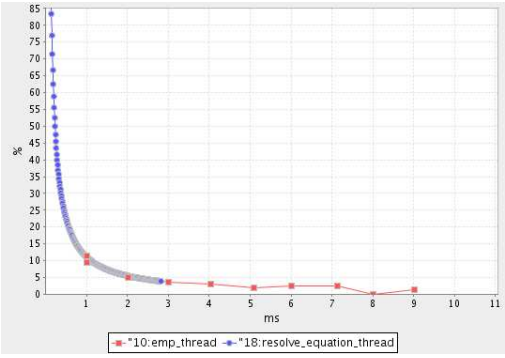


Fig. 6. Dual Core CPU cost

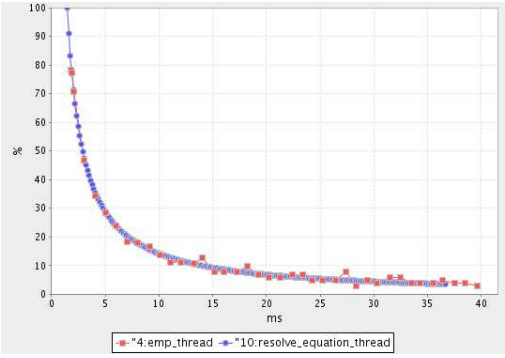


Fig. 7. Via Epia 1000 CPU cost

The empirical approach for the dual core figure stops at 12% load. This means that the manager cannot stress the remote equipment to a big load. This is due to the fact that the dual core is too powerful for the manager to be loaded. For the three figures we see that the empirical and the α / β approaches produce the same results. The latter is far faster since all the curves are produced from two reference measures.

E. Monitoring

Once the service provider has determined which data he wants to monitor and at which rate he wants to get information, he sets these values on the monitoring management panel. Figure 9 shows an example of remote monitoring panel. In this example the panel is targeted to a human administrator but it can be aimed at an automated system based on thresholds and alarms.

[2]All the code is available as opensource at http://cwiki.apache.org/FELIX/index.html, and http://mosgi.gforge.inria.fr/

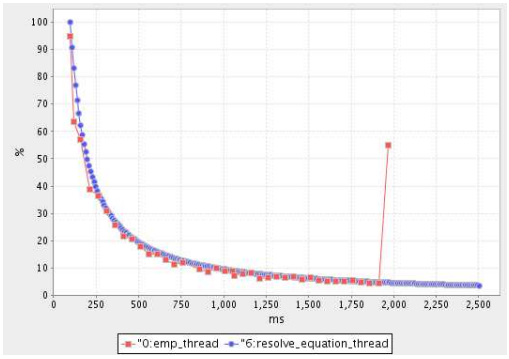


Fig. 8. LinkSys NLSU2 CPU cost

The monitoring console shows 2 periods (PI, PII). They display monitoring curves for 4 probes (c1, c2, c3, c4). From upper to lower curve :

- The c1 curve is the OSmemFree probe,
- The c2 curve is the OSmemBuffer probe,
- The c3 curve is he Cpu probe,
- The c4 curve is the OSmemCached probe.

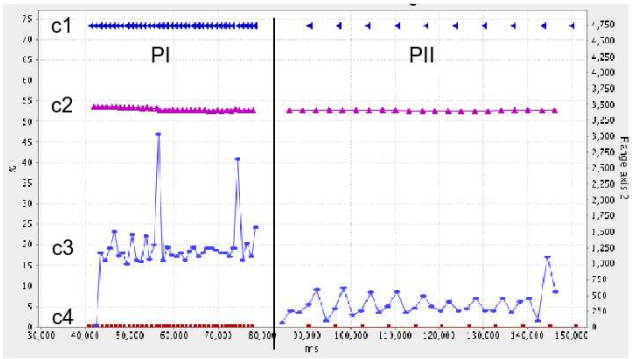


Fig. 9. Slug monitoring platform

In PI all probes request periods are set to 1s. So our framework predicts a CPU load for c1 to c4 respectively of 4.4%, 4.2%, 4.6% and 4.1%. We can see that the Cpu probe returns an average value of about 17.5% which is about the sum of the c1 to c4 probes cost. In PII we want to specify a request period for Cpu probe (c3) of 3s and for OSmemFree (c1) of 2s which, according to our framework, generates a CPU load of 3.6% total. So if we do not want to generate more than 5% CPU load with our management plan, we set request periods for the 2 remaining probes in order to generate a maximum of 1.4% CPU load. In the figure we choose *i.e* 0.7% for c2 and 0.7% for c4 the system calculates a resulting delay of 6069ms for c2 and 6623ms for c4.

A last use case of our system is to make benchmarking between systems. We use our system both on a Felix OSGi framework and on a Concierge OSGi framework. We observed that the mean response time of most probes are slightly faster on Concierge implementation than on Felix which confirms

that Concierge is optimized for small environments. These benchmarking issues are rather easy to set up and the results are directly observable with the monitoring platform.

Another use case we examined is to evaluate the quality of probe implementations. Our system can compare CPU consumption of similar probes in order to compare their performance.

VI. Conclusion

We present in this article a monitoring framework that aims at finding the balance between the frequency of remote queries and the resulting CPU load on the managed equipment. The monitoring process starts with an evaluation period where monitored system probes are queried in order to establish their characteristics. A probe characteristics is modeled with a curve that represent the induced load in function of the query pace. The curve has an $\frac{\alpha}{x} + \beta$ shape.

After having evaluated the curve a manager can elaborate a query plan. This plan fixes for each probe the query pace thus anticipating the resulting load induced by the management layers on the managed system. The overload is then due to other running applications. Of course one goal of the system is to guarantee that the management layers do not cost too much and that they stay below a certain limit.

This framework is aimed at home gateways; these run services and applications that should be remotely managed. Since those devices have quite limited resources, the management overload needs to be carefully tuned. The faster the management queries are made the faster the service provider can react to a perturbation, but the highest load he puts on the gateway.

References

[1] MUSE Project, "IST-507295 FP6," http://www.ist-muse.org/, 2004.

[2] Baskar Sridharan and Aditya Mathur, "Infrastructure for the Management of SmartHomes," Software Engineering Research Center Tech Report SERC-TR-177-P, January 2002.

[3] Radu State, Olivier Festor, and Isabelle Chrisment, "Context-Driven Access Control to SNMP MIB Objects in Multi-homed Environments," in *DSOM 2003, Self-Managing Distributed Systems*, vol. LNCS 2867, 2003.

[4] R. Levy, J. Nagarajarao, G. Pacifici, A. Spreitzer, A. Tantawi, and A. Youssef, "Performance management for cluster based web services," in *IFIP/IEEE Eighth International Symposium on Integrated Network Management*, 2003.

[5] Markus Debusmann, M. Schmid, and M. Kröger, "Generic Performance Instrumentation of EJB Applications for Service-Level Management," in *NOMS 2002, Florence, Italy*, ser. Lecture Notes in Computer Science, M. Brunner and A. Keller, Eds. Springer, April 2002.

[6] Sujay Parekh, Kevin Rose, Joseph Hellerstein, Sam Lightstone, Matthew Huras, and Victor Chang, "Managing the Perfomance Impact of Administrative Utilities," in *DSOM 2003, Self-Managing Distributed Systems*, vol. LNCS 2867, 2003, http://www.research.ibm.com/PM/RC22864.pdf. [Online]. Available: http://www.research.ibm.com/PM/

[7] H. Kreger, "Java management extensions for application management," *IBM Systems Journal*, vol. 40, no. 1, pp. 104–129, 2001.

[8] M. Chung, "Using jconsole to monitor applications," Sun Whitepaper, December 2004. [Online]. Available: http://java.sun.com/developer/technicalArticles/J2SE/jconsole.html

[9] Abdelkader Lahmadi, Laurent Andrey, and Olivier Festor, "Performances et resistance au facteur d'échelle d'un agent de supervision basé sur JMX : Méthodologie et premiers résultats," in *GRES*, 2005.

[10] Eric Fleury and Stéphane Frénot, "Building a JMX management interface inside OSGi," Inria RR-5025, Tech. Rep., 2003.

[11] Yvan Royon, Stéphane Frénot, and Frédéric Le Mouël, "Virtualization of Service Gateways in Multi-provider Environments," Component-Based Software Engineering, 2006.

[12] Yvan Royon and Stéphane Frénot, "Architecture de gestion de passerelles de services," in *GRES*, 2006.

[13] OSGi Alliance, "http://www.osgi.org/."

[14] Dave Marples and Peter Kriens, "The Open Services Gateway Initiative: an Introductory Overview," IEEE Communications Magazine, December 2001.

[15] Apache Software Foundation, "Felix OSGi R4 Service Platform implementation," http://felix.apache.org/. [Online]. Available: http://svn.apache.org/repos/asf/incubator/felix/

2.3 Project SOSGi[Parrend et al. 2007]

Pierre Parrend and Stéphane Frénot.

Security Benchmarks of OSGi Platforms : Towards Hardened OSGi.
To be published in Software Practice and Experience, 2009

SOFTWARE—PRACTICE AND EXPERIENCE
Softw. Pract. Exper. 2000; **00**:1–7 *Prepared using* speauth.cls *[Version: 2002/09/23 v2.2]*

Security Benchmarks of OSGi Platforms: Toward Hardened OSGi

P. Parrend[†] and S. Frenot[*,‡]

INRIA Ares/CITI, INSA-Lyon, F-69621 France

SUMMARY

OSGi Platforms are Extensible Component Platforms, *i.e.* they support the dynamic and transparent installation of components that are provided by third party providers at runtime. This features makes systems that are built using OSGi extensible and adaptable, but opens an dangerous attack vector that has not been considered as such until recently.

Performing a security benchmark of the OSGi platform is therefore necessary to gather knowledge related to the weaknesses it introduces as well as to propose enhancements. A suitable *Vulnerability Pattern* is defined. The attacks that can be performed through malicious OSGi components are identified. Quantitative analysis is then performed so as to characterize the origin of the vulnerabilities and the target and consequences of attacks. Assessment of the security status of the various implementations of the OSGi Platform and of existing security mechanisms is done through a metric we introduce, the *Protection Rate*.

Based on these benchmarks, OSGi-specific security enhancements are identified and evaluated. Recommendations are given. Evaluation is performed through the Protection Rate metric and through performance analysis. Lastly, further requirements for building secure OSGi Platforms are identified.

KEY WORDS: Software Security Assurance, Software Vulnerabilities, Security Benchmark, OSGi, Component Platform, Dependability

1. Introduction

The OSGi Platform is a Java-based Component Platform. It enables management of several applications on the same Virtual Machine as well as runtime installation, starting, stopping

[*]Correspondence to: INRIA Ares/CITI, INSA-Lyon, F-69621 France. Phone: (+33) 4 7243 6415. Fax: (+33) 4 7243 6227.
[†]E-mail: pierre.parrend@insa-lyon.fr
[‡]E-mail: stephane.frenot@insa-lyon.fr

Received 11 February 2008
Revised 11 February 2008

and uninstallation of the so-called *bundles*. These features characterize what we call *Extensible Component Platforms*. Other examples are Java MIDP [18], Android which is built on a Java/Linux stack, and the .Net Platform by Microsoft.

Runtime extensibility provides a radical improvement in flexibility and management of applications. However, it also opens a brand new attack vector: the dynamic and transparent installation of bundles that may be provided by unknown or at least uncontrolled providers. In particular, bundles are installed automatically to provide all dependencies of an application that is being installed. This attack vector can be exploited in two ways. Either malicious developers act in the development team of the bundle provider or uncontrolled providers publish malicious bundles that are installed dynamically on the victim platform.

So as to improve the security status of existing OSGi Platforms, a set of requirements are identified. First, a better knowledge of the security risks that are bound with OSGi is necessary. Few works have been done so far on the subject. Secondly, performing a security benchmarking of the open source implementations of the OSGi platform is mandatory to make developers aware of the actual weaknesses of the development environment they are working on. The relative importance of OSGi-specific weaknesses and more widespread weaknesses such as those that are implied by the underlying Java Virtual Machine is of particular interest, since the latter are highly likely to concern other Java Platforms. Thirdly, recommendations for building more secure implementations of the OSGi Platform are required so as to patch the identified weaknesses and not let them open for aggressions.

The remaining of the paper is organized as follows. Section 2 defines the OSGi Platform and gives an overview of its security strength and weaknesses. Section 3 presents the catalog of vulnerabilities that we identified in the OSGi Platform, as well as security benchmarking results for Open Source implementations. Based on these data, recommendations for building Hardened OSGi Platforms are given and evaluated in Section 4. Section 5 concludes this work.

2. The OSGi Platform

Performing security benchmarking for a given system implies that this system is properly understood, and that its key security properties are identified. Core elements of OSGi-based systems, the platform itself and the *bundles* (name of the OSGi components) are therefore first defined. Next, these definitions are formalized as taxonomies to support systematic analysis. Thirdly, an overview of known strengths and weaknesses of the OSGi Platform is provided.

2.1. Structure

The structure of OSGi-based systems is defined in OSGi Specifications Release 4.1 [25]. The specifications presents the architecture of the OSGi Platform and defines the format of bundles, in particular what relates to metadata.

The Platform

The OSGi Platform is compound of four layers that are running on top of a Java Virtual Machine: the Security Layer, the Module Layer, the Life-Cycle Layer and the Service Layer (Figure 1).

The Java Virtual Machine can be any JVM that provides an over-set of the Java Connected Device Configuration (CDC) Foundation Profile. Standard JVM from 1.3 upwards are supported.

The Security Layer supports the validation of digital signature of bundles and enforcement of execution permissions inside the OSGi Platform. Digital signature of OSGi bundles is very similar to the digital signature of Jar files. However, verification criteria are stronger, so specific verification tools have to be used [29].

The Module Layer plays the role of dependency resolver between the bundles inside the platform. Actually, each bundle is executed in a specific ClassLoader, which enables class isolation between the various applications. So as to make the interactions between bundles possible, dependencies have to be explicitly defined, *i.e.* the name of the required packages are to be mentioned as metadata in the bundles. A bundle can be installed only if all its dependencies are resolved, *i.e.* if all required libraries are available. A specific type of bundle is also handle by the Module Layer: fragment bundles, which are 'slave' bundles that are used to provide configuration information or context-dependent code to a 'Host' bundle. They cannot be used independently.

The Life-Cycle Layer deals with the installation, start, stop, update and uninstallation of the bundles. These advanced management features are in the core of the power of the OSGi Platform. It enables to load new bundles at runtime without disturbing the execution of already installed ones.

The Service Layer provides the support of Service Oriented Programming [4]. Bundle can publish services in a common *Context* under the form of Java Interfaces. They are frequently augmented with specific properties that enable service search using the LDAP request format [16]. These services can be dynamically discovered and used by other bundles without requiring that dependencies are defined at the Module Level.

A Bundle Downloader bundle (often called 'Bundle Repository' because it handles remote repositories) completes the full support of the bundle life-cycle. It is defined as an OSGi Request for Comments document [27]. It performs discovery and download new bundles over the Internet. Metadata format for the Bundle Repositories are named OBR v2 (Open Bundle Repository v2).

The Bundles

OSGi Bundles are Java Archive (Jar) files [32] that are extended to support specific operations such as life-cycle management and dependency resolution. They are built of two main types of data: Meta-data and application code. Their structure is shown in Figure 2.

The Meta-data are defined in the **Manifest** file of the archive. Most important informations are the bundle symbolic name, its version number, the reference of the Activator class, and the list of imported and exported packages. The list of required and provided OSGi services is optional. The Activator class is a specific Java class that performs starting activities for the bundle, such as configuration and service handling. It contains two methods start and stop that are called when the bundle is started and stopped. The list of imported and exported

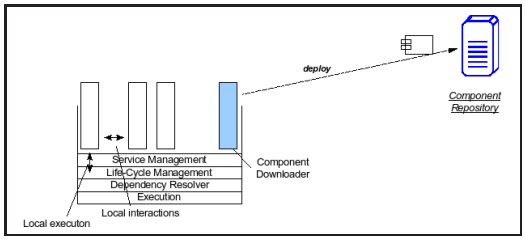


Figure 1. Platform Model for OSGi

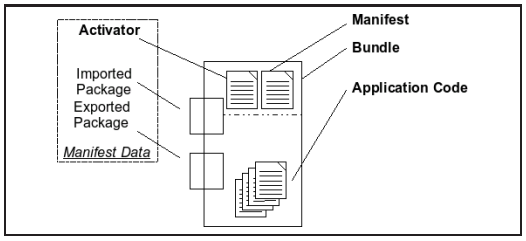


Figure 2. Component Model for OSGi

packages enables proper dependency resolution. All packages that are not exported are kept private to the bundle.

The application code is standard Java code. It can of course also contain native code as any Java application. It can be either used by the bundle only, made available as packages where all classes can be used by third party bundles, or made available as OSGi Services.

A bundle can be in one the following life-cycle phase: installed (be manageable from the platform), resolved (installed and with all dependencies resolved, after complete installation of after stopping), active (after its start), or not visible. Through the shared BundleContext, bundles can be managed by all other bundles in the Platform unless specific management permissions are set.

2.2. Taxonomies for OSGi-based Systems

Systematic security analysis of a system implies that a model of this system is available. We therefore define two taxonomies that represent the core elements of an OSGi-based system, the platform and the bundle.

The taxonomy of the elements of an OSGi Platform is presented in Figure 3. It is subdivided in three subsystems: the Operating System, the JVM and the OSGi Platform. The JVM can be subdivided between the Runtime, which is the execution engine, and the Classpath which is built up by the Standard API classes. The JVM is specified by [19]. The OSGi Platform is built up by the Module Layer, the Life-Cycle Layer and the Service Layer. The security layer is split between these three internal layers.

The taxonomy of the elements of OSGi Bundle Elements is presented in Figure 4. These elements can be classified into two main categories: intra-bundle structure and inter-bundle interactions. The intra-bundle structure is compound of the following items: the bundle archive, the Manifest file, the Activator Class, Native Code, the Java Language, the Java API calls, and OSGi API calls. Inter-bundle specific interactions can be performed through OSGi services or OSGi Fragments. Interactions through package export/import are not considered explicitly since they can be considered either as related to Meta-data (Manifest file) or as standard Java calls.

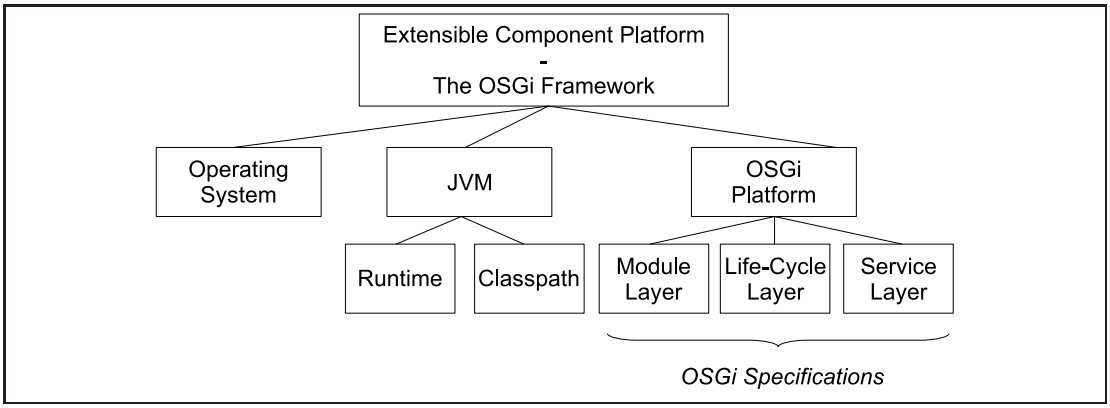


Figure 3. Taxonomy of the Elements of an OSGi Platform

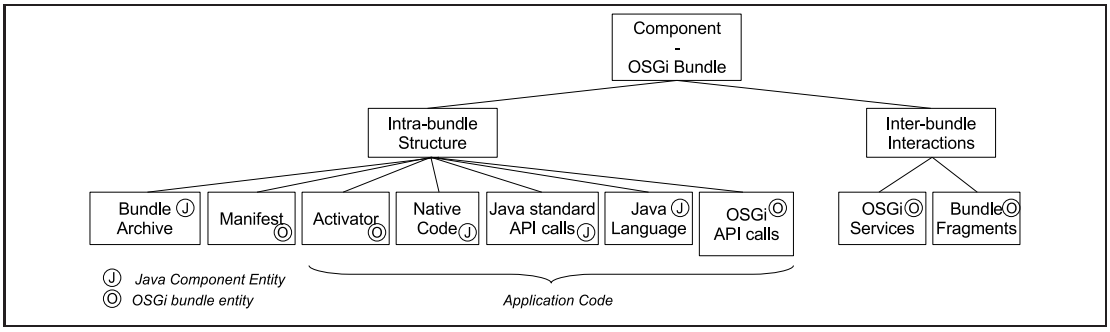


Figure 4. Taxonomy of the Elements of an OSGi Bundle

2.3. Security Strengths and Weaknesses of OSGi Platforms

The OSGi Platform is deemed to be highly secure by its proponents. Actually, it has several built-in features that makes it a serious candidate for developing secure execution platforms. However, the state of current implementations shows that this matter is still poorly addressed in spite of the strong security requirements of the common OSGi use cases.

Why is OSGi potentially secure

The two major arguments in favor of the strong potential of OSGi as a secure execution platform are the following. First, OSGi is executed on top of a Java Virtual Machine. Secondly, it has been designed to support a proper namespace isolation between bundles.

The Java language and Virtual Machine is one of the execution environment that has been subject to extensive tests and validation since its publication (see for instance [30, 11, 17]). It has been originally designed to support the safe execution of fully untrusted code, such as Web Applets [10]. Four mechanisms support this feature [13]: Type Safety of the language, Garbage Collection which prevents user-defined memory management, which is error prone, Bytecode verification to ensure that executed programs are compliant with the language specification, and Sandboxing, which prevents the access to sensible resources such as the network, or the file system. Two additional mechanisms are meant to support secure modular programming: secure ClassLoaders, which make it possible to load several modules that can not interact with each other, and customizable permissions, which enable flexible sandboxing by allowing only necessary rights to the code that needs it [14].

On top of this sound basis, the OSGi Platform extends the component-based approach to permissions. First, it provides a clean implementation of ClassLoaders, which prevents the application designer from misusing or by-passing them. Secondly, it defines specific permissions, that enable to fully control the interactions between the bundles: dependencies at the package level and at the service level can be authorized or prevented. Management capability from one bundle to the others can also be restricted.

This clean management model makes it possible to claim that the OSGi Platform is security aware, and that it provides advanced security features that are available on few others. However, we will show that these security efforts in the specifications are not sufficient to provide a secure programming model for a multi-applications platform.

Why is OSGi currently not secure

As presented in the taxonomy for the elements of an OSGi Platform, security flaws can originate in three parts: the Operating System, the Java Virtual Machine and the OSGi Platform itself.

Flaws in Operating Systems are out of scope of this study. A reference for the Unix System can for instance found in [3]. It will therefore not be discussed further, all the more as the Java Sandbox is considered since the inception of Applets as a good protection against malicious code. Unless it is explicitly authorized, no bundle can have access to the underlying system.

Flaws in the Java Language and Virtual Machine can be divided in two categories. The first one is built up by raw flaws that enable code to harm the system. The second one is related to the security model of Java. This latter has originally not been designed for multi-applications systems and proves to fall short providing strong isolation between mutually untrusted applications.

Several unexpected and somewhat unlogical behavior are identified in the Java Language [6, 7].This can lead to unsafe behavior, difficult to track infinite loops and memory leaks. Moreover, several failures are reported to take place in JVMs when they are under heavy workload [8]. In particular, optimizations such as JIT (Just-in-Time) compilers or Garbage Collectors are an important source of weaknesses. Both of these failure types can be used to launch Denial-of-Service (DoS) attacks against any Java execution Platform.

Java has been designed for secure execution of single application, and current security mechanisms for secure modular systems are an extension of the single-application mechanisms. In particular, namespace isolation is supported thanks to the ClassLoading mechanism, but

resource isolation, *i.e.* isolation of available CPU and memory is only supported in specific implementations of the Java Virtual Machine, such as the Multi-User Virtual Machine [9]. The fact that this latter has not yet been widely deployed shows that it is not yet mature enough to be used in production systems. Consequently, requirements for proper multi-applications Java-based system are so far only partially addressed.

Flaws in the OSGi Platform have so far been hardly studied. Though the platform has been subject to an important security effort, the flexibility it brings into systems do not come without new risks. The actual security model that is currently supported therefore needs to be clarified.

As an execution platform that targets networked environments, the OSGi platform has been subject to an important effort for enhancing its security. The specifications introduce a specific security layer to support the verification of the digital signature of bundles and to manage OSGi specific execution permissions [25]. Moreover, recommendations for building secure OSGi-based systems are detailed in a dedicated OSGi Request for Comments [26]. Nonetheless, these specifications are not systematically built-in into available implementations. In particular and to the best of our knowledge, no implementation of the security layer is so far available in open-source implementations of the Platform. Differences between default Java specifications and OSGi ones are not yet considered in mainstream projects, for instance what concerns the criteria of validation of digital signature of bundles [29][†].

Specific security implications of OSGi features also have been so far neglected. In particular, the bundle life-cycle support and the Bundle Repository tool enable to install in a dynamic and transparent manner bundles that are provided by possibly unknown developers through public repositories. This flexilibity is the very enhancement that is brought by the OSGi Platform, but it opens a new and easily exploitable attack vector. Understanding of the risks that are implied is necessary to adapt the use cases of the OSGi Platforms to the security status that is actually provided. This is what we intend to do in this work.

The security model that is currently supported by OSGi-based systems is the following. A platform must only accept signed bundles if it wants to have any control on the origin of the executed code. The bundle signer is thus liable for any actions of the bundles it provides. Since the bundles are signed but never encrypted, no confidential information should be transmitted under this form. Any system that requires both the flexibility of OSGi bundle management and more complete security guarantees must use ad-hoc security mechanisms.

OSGi-based systems can be divided into two main elements: the platform itself, and the bundles. Though it is designed with several efficient security mechanisms embedded in it, the Platform is likely to contain vulnerabilities as any software system do. The bundles can contain code that exploits these vulnerabilities to harm the system where they are installed.

As the use of OSGi increases in environments such as Smart Homes or Web Servers, the potential security flaws may soon be an efficient way of performing costly attacks. A better knowledge of existing flaws in the current specifications and open source implementations of the OSGi Platform is therefore required.

[†]http://sfelix.gforge.inria.fr/

3. Security Benchmarking of OSGi Platforms

Security Benchmark of existing OSGi Platforms is required to gather knowledge that can be exploited to enhance their security status. Requirements are twofold: build a catalog of the vulnerabilities that exist in the platform, and perform a benchmarking of the open source OSGi Platforms against the vulnerabilities of the catalog. We restrict ourself to Open Source implementations of the OSGi Platform, because our goal is to help enhancing them, and not to criticize commercial products. These latter are often tuned for specific environments and benchmarks should take these constraints into account, whereas open source projects do not make such assumptions. Moreover, the access to the code of such platforms is not necessarily free or legal.

Building such a catalog requires a Vulnerability Pattern to be available, that standardizes the data that is gathered for each vulnerability and supports security benchmarking for the considered platform type, *i.e.* the OSGi Platforms. This Pattern is first introduced. Next, malicious bundles that threaten OSGi elements are listed. Lastly, security benchmarks of the most current implementations of the OSGi platform are presented.

3.1. The Vulnerability Pattern for Security Benchmarking of OSGi Platforms

Gathering informations relative to vulnerabilities implies that a common pattern is available that defines the type of data that is necessary to handle them.

Existing Patterns and Languages for vulnerability documentation are first discussed. Since none is available that supports security benchmarking for OSGi Platforms, a dedicated *Vulnerability Pattern* is introduced. Its use is highlighted through an example.

Patterns and Languages for Documenting Vulnerabilities

If the use of pattern in Software Engineering is a commonplace since the 'Band of Four' book, the application of this approach to security engineering is more recent [31]. Patterns and languages for documenting vulnerabilities can be parted into three categories: informative patterns, vulnerability and exposure patterns for operational support, and languages for vulnerability assessment and resolution.

Informative Patterns are meant for broad disclosure of vulnerability data. They provide the minimal information that is required by systems administrators to perform security update. Example of such patterns are the Rocky Heckman pattern[‡], the CERT (Computer Emergency Response Team) pattern and the CIAC[§] (US Department of Energy) pattern.

The second type of Patterns is defined to enable Vulnerability and Exposure Description for operational support. This type is also known as *VEDEF* (Vulnerability and Exposure Description and Exchange Format). They are machine-readable descriptions of the vulnerabilities and of the remediation process. So as to be exploitable, they should contain the following data fields [2]: description of the platform(s) that is(are) affected, description

[‡]http://www.rockyh.net/
[§]http://www.ciac.org/ciac/index.html

of the nature of the problem, description of the likely impact if the Vulnerability and/or Exploit were triggered, available means of remediation, disclosure restrictions. One of the de facto standard is the MITRE CVE[¶] (Common Vulnerability and Exposures) Pattern. Other examples are the XML Common Format for Vulnerability Advisories, by the EISPP [12], the Common Advisory Interchange Format (CAIF), by the RUSCERT[‖], and the Advisory and Notification Markup Language (ANML), by OpenSec. More specific propositions, for instance the Application Vulnerability Description Language (AVDL), by OASIS, are defined for web applications.

The VEDEF Vulnerability Pattern type contains several useful informations for performing security benchmarking of OSGi Platform, such as the description of the vulnerability, of the affected platforms and of the means of remediation. However, no existing pattern supports component-platform specific properties such as Life-Cycle information or benchmarking specific information such as reference to attack code implementation.

The third type of Patterns is defined to supported both the assessment and the remediation processes. One example is the OVAL Language (Open Vulnerability and Assessment Languages) [21]. OVAL is very similar to our proposition insofar it supports penetration testing. The main difference is that OVAL is used as a support in a series of security assessment and remediation tools, whereas our *Vulnerability Pattern* only support assessment and analysis, and is not integrated in tools.

The Vulnerability Pattern for Security Benchmarking of OSGi Platform

It falls in the category of vulnerability and exposure description for operational support, with extension to take penetration testing as well as component Life-Cycle into account.

It is represented in XML format, and contains following sections:

- Vulnerability Reference, to provide the unique identifier of the vulnerability, and a proper classification according to the taxonomies that are introduced in Sections 2.2 and 3.3,
- Vulnerability Description, to enable the detailed presentation of the behavior of the vulnerability, as in the Morbray pattern [23],
- Protections, being actual or potential ones that may be efficient,
- Vulnerability Implementation, the development status of the proof-of-concept malicious bundles.

The benefits of this Vulnerability Pattern are the following. It is the only one that considers informations that are related to the Life-Cycle of components, which is actually surprising with regard to the development of COTS-based systems. It includes the reference to one or several proof-of-concept implementations of the malicious bundles. It supports the elicitation not only of available security mechanisms, but also of potential ones, so as to support the identification of required development efforts in the domain of security protections when a specific platform is benchmarked. Lastly, specific values are predefined for each field of the 'Vulnerability Reference' section, according to the taxonomies that are defined in this study. This provides a common language for describing vulnerabilities in a coherent manner.

[¶] http://cve.mitre.org/
[‖] http://cert.uni-stuttgart.de/files/caif/requirements/split/requirements.html

The limitations of the Pattern are the following. It could be integrated into existing patterns, such as the OVAL one, as a platform-specific extension. This would make its integration with security management tools using OVAL possible, but would introduce an important development requirement to adapt the tools to this OSGi-specific extension. This is clearly out of the scope of this paper.

Example

So as to highlight the role of the defined Vulnerability Pattern, we now present an example of vulnerability: the 'Management Utility Freezing - Infinite Loop' vulnerability, which is given in Listings VII page 29 and VIII page 30.

This vulnerability consists in the presence of an infinite loop in the activator of a given Bundle, which causes the platform management tool (often an OSGi shell) to freeze. The presence of infinite loops as a vulnerability is given by Blooch in 'Java Puzzlers - Traps, Pitfalls and Corner Cases', puzzlers 26 to 33 [7]. The matching Pattern is first given, and then explained.

The 'Management Utility Freezing - Infinite Loop' is referenced under the identifier 'mb.osgi.4' ('malicious bundle catalog - originates in the OSGi Platform itself - number 4). This vulnerability is an extension of the 'Infinite Loop in Method Call' one. It has been identified in the frame of the research project 'Malicious Bundles' of the INRIA Ares Team.

The location of the malicious code that performs the attack is the Bundle Activator. Its source is the Life-Cycle Layer of the OSGi Platform, which is not robust against such a vulnerability. Its target is the Platform Management Utility, which can be either the OSGi shell or a graphical interface such as the Knopflerfish GUI. This vulnerability has a two-fold consequence: the method does not return, so that the caller also freezes; and the infinite loop consumes most of the available CPU, which causes the existing services to suffer from a serious performance breakdown.

This vulnerability is introduced during development, and exploited at bundle start time.

Related Vulnerability Patterns are 'Management Utility Freezing - Hanging Thread', that also targets the Management Utility, 'Infinite Loop in Method Call', 'CPU Load Injection', 'Stand-alone Infinite Loop' that have the same consequence of performance breakdown, and the 'Hanging Thread', that also freezes the calling thread.

No specific protection currently exists. Two potential solutions have been identified. The first consists in launching every Bundle Activator in a new Thread, so as not to block the caller if the activator hangs. The second solution would enable to prevent invalid algorithms to be executed: static code analysis techniques such as Proof Carrying Code or similar approaches can provide formal proves of code wellformedness.

Its reference implementation is available in the OSGi bundle named 'fr.inria.ares.-infiniteloopinmethodcall-0.1.jar', referenced on 2006-08-24. The test coverage is 10 %, since ten types of infinite loops have been identified and only one has been implemented. The test bundle has been tested on the following implementations of the OSGi platform: Felix, Equinox, Knopflerfish, and Concierge. The only robust Platform against this attack is our Hardened Felix Platform, which is a prototype meant to enhance the current Felix implementation. Hardened Felix is presented in Section 4.

3.2. Threatening OSGi Elements through Malicious Bundles

The various vulnerabilities that threaten OSGi-based system are now presented.

As any vulnerability list, this one does not pretend to be exhaustive, but intends to be as complete as possible. In particular, the identification is based on the taxonomies we presented in the Section 2.2. Analysis is performed in a systematic way, *i.e.* all elements of the OSGi Platform and of the bundles are carefully studied to identify security holes.

A classification is established according to the origin of the attack inside the malicious bundle: Bundle Archive, Bundle Manifest, Bundle Activator, Bundle Code (Native), Bundle Code (Java), Bundle Code (OSGi API), Bundle Fragments. For each vulnerability, its name, its identifier, a quick description as well as its consequences are given.

Full descriptions of the vulnerabilities according to the Vulnerability Pattern is available in the related Technical Report [28]. Proof-of concept implementation of the malicious bundles is available for most vulnerabilities under simple request to the authors. Features that can be protected by a Security Manager [14] are nonetheless considered as vulnerabilities, essentially because the important performance overhead forces many developers not to use the Security Manager and associated Java Permissions. The second reason is that specified Permissions [33] are often partially implemented, in particular in the Sun JVM, which makes them quite useless.

Bundle Archive

Vulnerabilities that originate in the Bundle Archive are due to flaws in the structure of the archive. They are not bound with the content of the archive, and are therefore not specifically bound to the Java world.

Invalid Digital Signature Validation (mb.archive.1) a bundle which signature is NOT compliant to the OSGi R4 Digital Signature is installed on the platform; bundles with lacking or additional malicious classes can then be installed,

Big Component Installer (mb.archive.2) remote installation of a bundle which size is bigger than the available device memory; this results in rapid disk space exhaustion,

Decompression Bomb (mb.archive.3) the Bundle Archive is a decompression Bomb (either a huge file made of identical bytes, or a recursive archive); this leads to memory exhaustion.

Bundle Manifest

Vulnerabilities that originate in the Bundle Manifest are due to flaws in the expressed Metadata. They are bound either to the way the JVM handles the Manifest, or to OSGi-defined properties.

Duplicate Package Import (mb.osgi.1) a package is imported twice (or more) according to manifest attribute 'Import-Package'. In the Felix and Knopflerfish OSGi implementations, the bundle can not be installed. This is due to the OSGi Platform.

Excessive Size of Manifest File (mb.osgi.2) a bundle with a huge number of (similar) package imports (more than 1 MByte); this behavior originates in the Virtual Machine

Table I. Example of Malicious Code in OSGi bundle: Infinite Loop in Bundle Activator

```
public class InfiniteStartupLoopActivator implements BundleActivator{
        public void start(BundleContext context){
                System.out.println("Bundle InfiniteStartupLoop started");
                while(1==1){}
        }
        public void stop(BundleContext context){
                System.out.println("Bundle InfiniteStartupLoop stopped");
        }
}
```

(in particular, SUN JVM and JamVM are concerned), and leads to a temporary freezing of the Platform when the Manifest is loaded into memory,

Erroneous values of Manifest attributes (mb.osgi.3) a bundle that provides false meta-data, for instance an non existent bundle update location. This is bound with OSGi Meta-data.

Bundle Activator

Vulnerabilities that originate in the Bundle Activator are bound to OSGi bundle starting process.

Management Utility Freezing - Infinite Loop (mb.osgi.4) an infinite loop is executed in the Bundle Activator (see Listing I); this freezes the process that has launched the starting of the bundle, and consumes most of the available CPU,

Management Utility Freezing - Thread Hanging (mb.osgi.5) a hanging thread in the Bundle Activator makes the management utility freeze. Already installed bundles are not impacted.

*Bundle Code (Native)*** Vulnerabilities that originate in Native Code are bound to the possibility that exists in the Java Virtual Machine to execute code outside of the JVM. They are therefore not specific to the Java world.

Runtime.exec.kill (mb.native.1) a bundle that stops the execution platform through an OS call,

CPU Load Injection (mb.native.2) a malicious bundle that consumes an arbitrary amount (up to 98%) of the host CPU. Other processes on the platform experience an important loss of performance.

**These Bundles are specifically targeted to the OSGi Platform. All native code attacks against the Operating System or other applications are not considered here.

Table II. Example of Malicious Code in OSGi bundle: Recursive Thread Creation

```
public class Stopper extends Thread{
    Stopper(int id, byte[] payload)
    {
        this.id=id;
        this.payload = payload;
    }
    public void run()
    {
        System.out.println("Stopper id: "+id);
        Stopper tt = new Stopper(++id, payload);
        tt.start();
        Stopper tt2 = new Stopper(++id, payload);
        tt2.start();
        Stopper tt3 = new Stopper(++id, payload);
        tt3.start();
    }
}
```

Bundle Code (Java API)

Vulnerabilities that originate in Java API are due to features that are provided through the Java Classpath, *i.e.* libraries that are provided along with the JVM.

System.exit (mb.java.1) a bundle that stops the platform by calling 'System.exit(0)',

Runtime.halt (mb.java.2) a bundle that stops the platform by calling 'Runtime.getRuntime.halt(0)',

Recursive Thread Creation (mb.java.3) The execution platform is brought to crash by the creation of an exponential number of threads (see Listing II),

Hanging Thread (mb.java.4) Thread that makes the calling entity hang through interlocking (service, or package),

Sleeping Bundle (mb.java.5) a malicious bundle that goes to sleep during a specified amount of time before having finished its job,

Big File Creator (mb.java.6) a malicious bundle that creates a big (relative to available resources) files to consume disk memory space,

Code Observer (mb.java.7) a component that observes the content of another one through reflection; code and hard-coded configurations can be spied,

Component Data Modifier (mb.java.8) a bundle that modifies the data (i.e. the value of the public static attributes of the classes) of another one through reflection,

Hidden Method Launcher (mb.java.9) a bundle that executes (through reflection) methods from classes that are not exported or provided as service. All classes that are referenced (directly or indirectly) as class attributes can be accessed. Only public methods can be invoked.

Table III. Example of Malicious Code in OSGi bundle: Memory Load Injection (with 'size' parameters on the same order of magnitude than the total memory)

```
private void stressMem(int size)
{
    System.out.println("Eating " + size + " bytes of memory");
    this.memEater = new byte[size];
    for (int i=0 ; i<size ; i++)
    {
       this.memEater[i] = 0;
    }
}
```

Bundle Code (Java Language)

Vulnerabilities that originate in Java Language are bound with language-level features, in particular Object Orientation. They can therefore be relevant to other Object-Oriented Languages, and some are general enough to concerns any programming language.

Memory Load Injection (mb.java.10) a malicious bundle that consumes most of available memory (see Listing III); if other processes are executed that require memory, MemoryErrors are caused. This vulnerability concerns any multi-process system without resource isolation.

Stand Alone Infinite Loop (mb.java.11) a void loop in a lonesome thread that consumes much of the available CPU. This vulnerability concerns any multi-process system without resource isolation.

Infinite Loop in Method Call (mb.java.12) an infinite loop is executed in a method call (at class use, package use); this vulnerability concerns any programming language.

Exponential Object Creation (mb.java.13) Objects are created in an exponential way, and force the call to abort with a StackOverflowError. This vulnerability as such is specific to Object-Oriented languages, but variants can be built that use procedure recursions in any programming language.

Bundle Code (OSGi API)

Vulnerabilities that originate in the OSGi API are due to features that are provided by the OSGi Platform.

Launch a Hidden Bundle (mb.osgi.6) a bundle that launches another bundle it contains (the contained bundle could be masqueraded as a 'MyFile.java' file); any program can therefore be hidden.

Pirate Bundle Manager (mb.osgi.7) a bundle that manages others without being requested to do so (for instance, stops, starts or uninstalls the victim bundle) (see Listing IV).

Table IV. Bundle that launches a Bundle that is hidden inside it.

```
public void start(BundleContext context)
{
    try
    {
        //get data for the created bundle
        String fileName = "bigflowerpirat-1.1.jar";
        byte[] buffer = this.getBundleResourceData("/"+fileName);
        //create a new data file with loaded data
        File file = this.newDataFile(fileName, buffer, context);
        //install new bundle
        String location = file.getPath();
        System.out.println(location);
        Bundle b = context.installBundle("file://"+location);
        b.start();
    }
    catch(Exception e){e.printStackTrace();}
    System.out.println("Malicious BundleLoader started");
}
```

Zombie Data (mb.osgi.8) Data stored in the local OSGi data store are not deleted when the related bundle is uninstalled. It thus becomes unavailable and consumes disks space (especially on resource constraint devices). This is due to the implementation of the OSGi Platform.

OSGi Services and Bundle Fragments

Vulnerabilities that originate in the OSGi API are due to the specific type of interactions that exist inside the OSGi Platform. Those that are due to the Service-Oriented Programming (SOP) [4] paradigm are relevant to any SOP Platforms.

Cycle Between Services (mb.osgi.9) a cycle exists in the services call; this vulnerability is related to SOP,

Numerous Service Registration (mb.osgi.10) registration of a high number of (possibly identical) services through an loop; this vulnerability is related to SOP,

Freezing Numerous Service Registration (mb.osgi.11) registration of a high number of (possibly identical) services through an loop, that make the whole platform freeze in the Concierge implementation; this vulnerability is related to SOP.

Execute Hidden Classes (mb.osgi.12) a fragment bundle exports a package from its Host that this latter do not intend to make visible. Other bundles can then execute the classes in this package,

Fragment Substitution (mb.osgi.13) a specific fragment bundle is replaced by another, which provides the same classes but with malicious implementation,

Access Protected Package through split Packages (mb.osgi.14) a package is built in the fragment that have the same name than a package in the host. All package-protected

classes and methods can then be accessed by the fragment, and thus exported to the framework.

This catalog of vulnerability build the knowledge base that is required so as to perform the benchmarking of OSGi Platforms. Availability of such a knowledge base is asserted in particular in the NIST Samate project [5].

3.3. Security Benchmarking for Open Source Implementations of the OSGi Platform

Based on this vulnerability catalog, it is now possible to assess the security status of most widespread open-source implementations of the OSGi Platform. First, the origin of the vulnerabilities are analyzed to identify the type of intrusion techniques that are exploited and the relative responsibilities of the Java and OSGi environments. Next, targets and consequences of the attacks are detailed to enable the focus on the more serious attacks when protections are developed. Lastly, quantitative assessment and comparison of the various implementations of the OSGi platform is performed. The Protection Rate metric is introduced to support it.

Analyses are performed through automated parsing of the XML representation of the Vulnerability Patterns. Life-Cycle related informations are not considered further here, though they are available. Actually, they do not bring in information that is of outstanding interest for the benchmarking analysis.

Origin of the Vulnerabilities

Protecting a system against the vulnerabilities it contains imply that their causes and their relative importance are known. Two main aspects are to be considered: the part of the system where the vulnerabilities are located, and the type of intrusion technique that is used to exploit them.

A raw partition of the system can be made between the Java Virtual Machine and the OSGi Platform. This distinction enables to identify the relative liability of each part and to know which amount of the vulnerabilities is specifically due to the OSGi Platform. Out of the thirty-two (32) vulnerabilities that are presented in the catalog in Subsection 3.2, eighteen (18) are are implied by the JVM, *i.e.* 56 %. The number of vulnerabilities that are implied by the OSGi Platform is 14 out of 32, *i.e.* 44 %. This means that most of the vulnerabilities that are identified in this study are due to the JVM itself, and therefore that other Java-based Extensible Component Platforms such as MIDP also suffer from the same threats.

The relative importance of the intrusion techniques in an OSGi Platform are shown in Figure 5. The classification that is used as reference is the Neumann and Parker Classification [24], which defined nine categories of intrusion techniques that can be used against computing systems. Categories 3 to 8 are related to software attacks, other ones to non technological and hardware attacks. Category 3 concerns masquerading, which is not relevant here as far as no access control system is considered. Categories 4 to 8 are the following: 4) setting up subsequent misuse, 5) bypassing intended control, 6) active misuse of resources *i.e.* write or execution access to the system or resource (CPU, memory, disk space) consumption, 7) passive misuse of resources *i.e.* read-only access by a malicious bundle, and 8) misuse resulting from inaction.

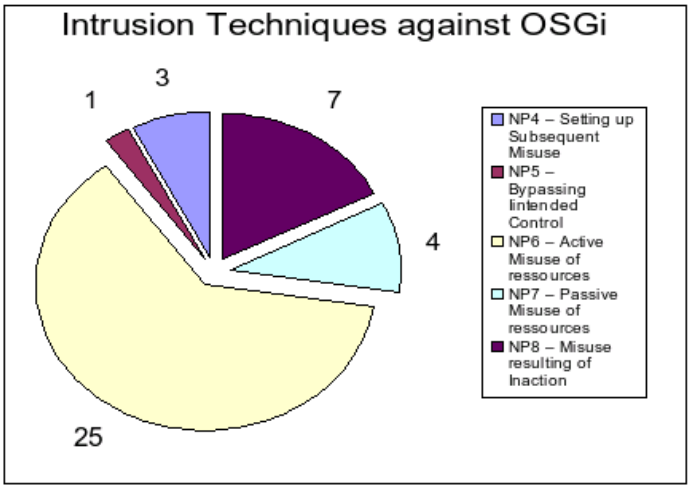


Figure 5. Relative Importance of Intrusion Techniques in an OSGi Platform

The distribution of the vulnerabilities in the Neumann and Parker categories is the following. One vulnerability can pertain to several categories.

Most vulnerabilities are *active misuse of resources* - 25 of them - which means that a great deal of the vulnerabilities are directly implied by actions that are performed by the malicious bundles. Preventing these actions through convenient access control mechanism would thus bring an important improvement in the security status of the OSGi Platform.

The number of *misuses resulting from inaction* is 7. These vulnerabilities are due to the fact that the OSGi Platform has not yet been designed to withstand attacks against it, and that therefore very few counter-measures are available.

The number of *passive misuse of resource* is 4, mainly undue read access. Since writing and reading access inside programs both occur through method calls, this kind of vulnerabilities can be prevented through the same mechanisms than active misuse, *i.e.* access control.

The number of vulnerabilities that consist in *setting up subsequent misuse* is three. They are related to bundle management and fragments and are solved by OSGi permissions.

The last type of vulnerability is *bypassing intended control*. The unique occurrence is due to the fact that JVM digital signature validation algorithm does not exactly match the OSGi specifications. A patch is presented in [29] and is available on the SFelix Web Site.

Vulnerabilities that originate in the JVM cause the majority or identified threats in the OSGi Platform. The OSGi specification itself is not free of such threats, as very few efforts have been so far done to make it secure. Though other types of weaknesses exist, most vulnerabilities are due to the absence of default access control mechanism, both at the Java and OSGi level.

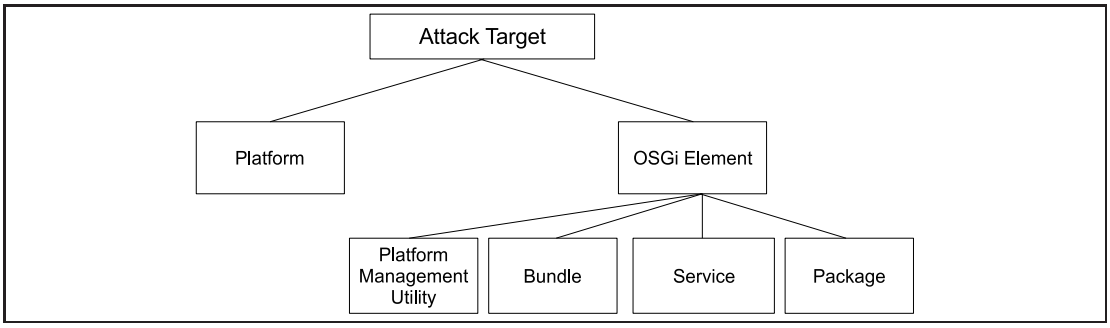


Figure 6. Taxonomy for Attack Targets in the OSGi Platform

Attacks: Targets and Consequences

Each vulnerability can be exploited to set up attacks against a platform. The objective of the analysis of their characteristics is to identify the ones that represent major threats to the system so as to express priorities in the process of securing the OSGi Platform. Two properties of the attacks are relevant to identify these priorities: the platform elements that are targeted, and the type of consequence they have.

Figure 6 shows the taxonomy of existing attack targets in OSGi-based Systems. The two main targets are the OSGi Platform itself, and the OSGi Elements. The Platform is the execution and management environment for the applications it contains. Consequently, attacking the Platform means that all of the applications that are running on it are impacted. The OSGi Elements are all the code that is executed inside the Platform. These elements can be impacted with a varying granularity: whole bundles can be attacked, single services, or packages. Attacks that target specific elements have impact on the element itself and to the elements that interact with it, but unrelated elements are not disturbed. A specific type of sensitive element is the Platform Management Utility, which is a single point of failure since its unavailability prevents the subsequent management of any new or already installed bundle.

The most serious attacks are those that target the whole platform. The relative importance of attacks that target specific elements depends on the number of interactions the victim element has with others, and of the application type: the unavailability of an entertainment service has not the same consequence that the unavailability of a health-care service.

The second characteristic of attacks is the consequence they have on the system under aggression. To reflect the specificity of the OSGi Platform, a dedicated taxonomy is introduced, as shown in Figure 7. As comparison with a mainstream classification, the Lindqvist Classification [20], is also given for reference.

The types of attack consequences for the OSGi Platform are unavailability, performance breakdown and undue access. Here, the target elements are not considered. One vulnerability can have several consequences. Unavailability is the consequence of 13 of the vulnerabilities, *i.e.* represents 40% of them. Performance breakdown concerns 11 vulnerabilities, *i.e.* 34%

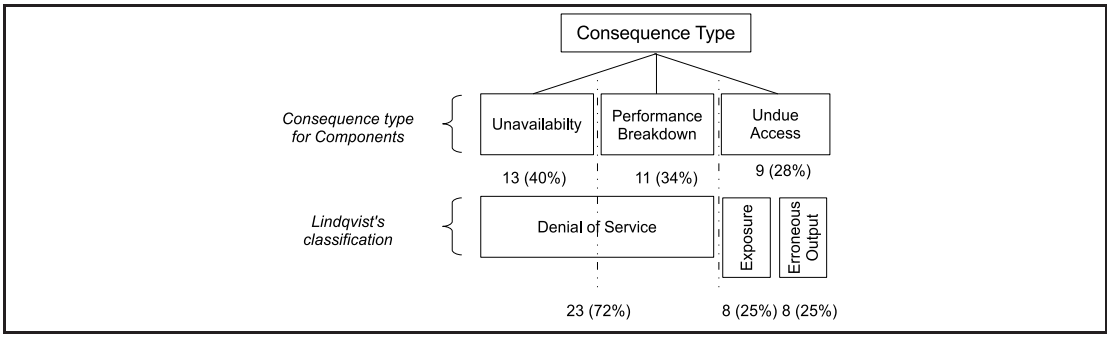


Figure 7. Taxonomy for Consequence Type in the OSGi Platform

of them. Undue access in read or write mode represents 9 vulnerabilities, *i.e.* 28%. Lindqvist classification establishes a distinction between the Exposure (*i.e.* read-only) and the Erroneous Output (*i.e.* read-write) undue access type. This reflects security properties of networks and is less relevant in our case: access to a given method of a bundle enables both read and write access, and does not depend on access control properties. It also considers all Denial of Service (DoS) attacks as a single type, whereas in the OSGi context unavailability and performance breakdown are clearly caused by different kinds of attacks.

The type of attack consequence that causes the most serious threats is highly dependent on the kind of application. For instance, Health-Care Systems are very sensitive to unavailability, whereas undue access is of the utmost importance in banking systems. In both cases, the wider the attack is, the most damages are caused. However, it is important to note that a recent trend in software security is that aggressions are targeted at very specific victims, and that a precise aggression to an element that is a single point of failure can cause even more damage than when the whole platform is attacked in a brute force manner.

Quantitative Security Assessment through the Protection Rate Metric

Security benchmarking requires that tools for quantitative security assessment are available so as to support evaluation of the Open Source implementations of the OSGi platform and comparison between them. To the best of our knowledge, no such metric is so far available.

A metric for fault-tolerant systems exists that addresses quantitative assessment and benchmarking: the *coverage* metric [1]. Its goal is to express the percentage of faults that are contained by fault-tolerance mechanisms. We therefore define the similar *Protection Rate* (PR) metric, which represents the percentage of the known vulnerabilities that are protected by a given security mechanism. Several implementations of the OSGi Platform are compared through their respective Protection Rate against the set of vulnerabilities we identified for the OSGi specifications.

The Protection Rate is based on the Attack Surface metric [15]. The Attack Surface is the number of functionalities and flaws that can be exploited to exploit a given system. In our

study, the attack surface is the total number of vulnerabilities that are identified in the OSGi Platform. Its value is 32.

The *Protection Rate* is defined as the quotient of the Attack Surface that is protected through any security mechanism and the Attack Surface of the reference system. It is also expressed as the complement of the quotient of the actual Attack Surface for the system to be evaluated and the Attack Surface of the reference system. In our case, the reference system is an idealized OSGi Platform that contains all vulnerabilities that are identified both in the specifications and in common implementations. The system to be evaluated is a concrete implementation of the OSGi Platform, possibly with some security mechanisms on. The Protection Rate can be expressed as:

$$PR = \left(1 - \frac{Attack\ Surface\ of\ the\ evaluated\ System}{Attack\ Surface\ of\ the\ Reference\ System}\right) * 100 \quad (1)$$

The Protection Rate metric enables to compare several similar platforms, for instance several implementations of the same specifications, by expressing the rate of protection that each of the platforms provides when compared to the set of the vulnerabilities that are identified for these specifications. The *Protection Rate* metric still has some limitations. It does only represent the rate of known vulnerabilities that are protected in a given flavour of the considered system. In particular, it does not reflect the relative importance of the vulnerabilities, *e.g.* according to the impact of the attacks that are made possible by them.

The Protection Rates for the various implementations of the OSGi Platform are given in Table V.

The default Felix implementation is robust against one of the identified attacks, out of thirty-two (32; $PR = 3,1\%$). The default Knopflerfish implementation is robust against one attack, out of thirty-one (31; digital signature of bundle is not supported; $PR = 3,2\%$). The default Equinox implementation is robust against 4 vulnerabilities, out of thirty-one (31; $PR = 13\%$), which is slightly better, but can not be considered as satisfactory. The default Concierge implementation is subject to all identified vulnerabilities. Since they do not support Bundle Fragments nor bundle signature, their Protection Rate is 0 / 28 = 0 %.

Java Permissions represent one solution that can be used to improve the security status of the OSGi Platform. They enable to protect it against thirteen attacks (13; $PR = 40\%$). Felix and Knopflerfish with Java Permissions have a Protection Rate of respectively 43 % and 45 %. Equinox with Java Permission has a Protection Rate of 53 %. Concierge with Java Permission has a Protection Rate of 36 %.

These results show that the various implementation are designed with rather limited security in mind. A good practice to enhance the security status of OSGi-based systems is to use Java Permissions. However, this peculiar mechanism is known to induce an important performance overhead during execution. For sensitive applications, a trade-off must so far be found between security and performance.

A Vulnerability Pattern for benchmarking of OSGi Platforms is defined. The vulnerabilities of the OSGi Platform are identified and described. Since a good half of them is due to the underlying Java Platform, this catalog can also be of interest for all designers of Java-based

Table V. Protection Rate for Mainstream OSGi Platforms

Platform Type	**# of Protected Flaws**	**# of Known Flaws**	**Protection Rate**
Concierge	0	28	0 %
Felix	1	32	3,1 %
Knopflerfish	1	31	3,2 %
Equinox	4	31	13 %
Java Permissions	13	32	41 %
Concierge with perms	10	28	36 %
Felix with perms	14	32	44 %
Knopflerfish with perms	14	31	45 %
Equinox with perms	17	31	55 %

component systems. Moreover, widespread open-source implementations of the OSGi Platform are benchmarked against this set of vulnerabilities.

An immediate requirement when vulnerabilities are identified is to make suitable security mechanisms available. Since this analysis is focused on the OSGi Platforms, OSGi specific protections are to be defined first. The creation of Java-specific security protections is mandatory for building secure production OSGi platforms, but they are out of scope of this study.

4. A Hardened OSGi Platform

The need for protecting OSGi Platforms urges us to define and evaluate a series of recommendations for developing hardened OSGi Platforms.

First, recommendations for building *Hardened OSGi* implementations are given, along with precise development requirements. Next, an evaluation of the prototype implementation, Hardened Felix, is performed through the Protection Rate metric that is defined in previous section. Then, the evaluation of the performances of the prototype implementation is presented to validate the usability of the recommendations.

4.1. Recommendations for Hardened OSGi Implementations

Recommendations for implementing hardened OSGi Platforms are made up of two parts. The first part contains recommendations that intend to complete the OSGi specifications. They are classified according to the layer structure of the OSGi Platform. The second part contains API and implementation requirements, *i.e.* modifications that have been necessary to implement the *Hardened OSGi* recommendations in our prototype.

Following modifications are to be taken into account at the specification level. They are presented according to the OSGi Platform Element that is concerned.

Module Layer The recommendation for a hardened OSGi Module Layer intends to make bundle dependency management more robust.

- **Bundle Dependency Resolution Process:** do not reject duplicate imports. Just ignore them; *OSGi R4 par. 3.5.4*. This prevents the attack 'Duplicate Package Import' (mb.osgi.1).

Life-Cycle Layer Recommendations for a hardened OSGi Life-Cycle Layer intend to protect the platform from installation of malicious or ill-formed bundles, as well as to guarantee that the uninstallation process is performed in a clean manner.

- **Bundle Download:** when a bundle download facility is available, the total size of the bundles to install should be checked immediately after the dependency resolution process. The bundles should be installed only if the required storage is available. This prevents the attack 'Big Component Installer' (mb.archive.2), without downloading the bundles on the platform. This security mechanism is to be used in conjunction with the maximum storage size mechanism in the context of OBR clients [27] for OSGi.
- **Bundle Installation Process:** a maximum storage size for bundle archives is set. Alternatively, a maximum storage size for all data stored on the local disk is set (*Bundle Archives and files created by the bundles*); *OSGi R4 par. 4.3.3*. This prevents the attack 'Big Component Installer' (mb.archive.2) of the OSGi Vulnerability Catalog.
- **Bundle Signature Validation Process:** the digital signature must be checked at installed time. It must not rely on the Java built-in validation mechanism, since this latter is not compliant with the OSGi R4 Specifications [29]; *OSGi R4, Paragraph 2.3*. This prevents the attack 'Invalid Digital Signature Validation' (mb.archive.1).
- **Bundle Start Process:** launch the Bundle Activator in a separate thread; *OSGi R4 par. 4.3.5*. This prevents the attacks 'Management Utility Freezing - Infinite Loop' (mb.osgi.4) and 'Management Utility Freezing - Hanging Thread' (mb.osgi.5).
- **Bundle Uninstallation Process:** remove the data on the local bundle filesystem when a bundle is uninstalled (and not when the platform is stopped); *OSGi R4 par. 4.3.8*. This prevents the attack 'Zombie Data' (mb.osgi.8).

Service Layer The recommendation for a hardened OSGi Service Layer intends to make the Secure Oriented Programming (SOP) features of the Platform more reliable.

- **OSGi Service Registration:** set a Platform Property that explicitly limits the number of registered services (default could be 50); *OSGi R4 par. 5.2.3*. This prevents the attacks 'Numerous Service Registration' (mb.osgi.10) and 'Freezing Numerous Service Registration' (mb.osgi.11).

To support this modifications of the OSGi platform implementations, following changes have been applied to the API. These modifications are required for storage size control.

- In the Class BundleContext, a method getAvailableStorage() is defined,

Table VI. Protection Rate for Hardened OSGi Platforms

Platform Type	# of Protected Flaws	# of Known Flaws	Protection Rate
Hardened OSGi Spec.	8	32	25 %
Hardened Felix	8	32	25 %
Hardened Felix with permissions	21	32	66 %
Hardened Equinox with perms (expected)	23	31	74 %

- A property osgi.storage.max is defined, that is set in the property configuration file of the OSGi framework.
- In the class org.osgi.service.obr.Resource, a method getSize() is defined. This method relies on the size entry of the bundle meta-data representation (usually a XML file).

Several attacks relative to the bundle archive, the bundle manifest, the bundle activator and OSGi API are prevented through this set of modifications. In particular, memory exhaustion due to the installation of big bundles or unclean uninstallation, or Denial of Service against the Platform Management Utility and the Service broker mechanism are prevented. Installation of maliciously modified bundles is also prevented.

These recommendations can be used in any OSGi implementation, so as to make it more robust in presence of malicious or simply ill-coded bundles. A prototype has been built based on the Felix Apache implementation[††] of the OSGi R.4 Platform to validate their usability.

4.2. Evaluation of Hardened Felix: Protection Rate

The benefit of the *Hardened OSGi* recommendations must now be evaluated. This is first done through the Protection Rate (PR) metric, which is extracted for the following configurations: absolute PR value, PR value for Hardened Felix, PR value for Hardened Felix with Java Permissions. Next, the relative importance of OSGi and Java vulnerabilities that stay unprotected within Hardened OSGi implementations are discussed. These data enable the comparison with other OSGi implementations.

The Protection Rates for Hardened OSGi Platforms are given in the Table VI. The Hardened OSGi Recommendations prevent eight of the identified vulnerabilities, out of thirty-two. The Protection Rate is therefore $PR = 25\%$. Our prototype, Hardened Felix, benefits of all of these protections, but one that is redundant with Felix features. Our prototype used with Java Permissions is robust against twenty-one (21) vulnerabilities out of thirty-two (32), *i.e.* 66 %.

[††]http://felix.apache.org

The relative importance of remaining Java and OSGi-specific vulnerabilities are discussed for the same configurations. Hardened Felix without Java Permissions has twenty-four (24) remaining vulnerabilities. Seventeen (17) of them are bound with the underlying JVM, *i.e.* 71 %. Seven (7) of them are bound with OSGi features, *i.e.* 29 %. However, these latter are due to OSGi features such as bundle management, OSGi services and fragments. Almost all of these can be protected through OSGi Permissions.

Hardened Felix with Java Permissions has ten (10) remaining vulnerabilities. Nine (9) of them are bound with the JVM, *i.e.* 90 %. One is bound with the OSGi Platform, *i.e.* 10 %.

One single OSGi vulnerability is not protected in Hardened OSGi: *Erroneous values of Manifest Attributes*. This can be prevented by automated generation of these attributes, which is often the case when Integrated Development Environments such as Eclipse or Maven are used. Moreover, this weakness may not be considered as an actual vulnerability but as a configuration error as any system can experience.

OSGi vulnerabilities can be considered to be fully patched by the Hardened OSGi recommendations, when suitable Java and OSGi Permissions are set. However, Java vulnerabilities keep being open and make OSGi Platforms at risk. The remaining major attacks that can occur in a Hardened Felix Platform with Java Permissions set are the following: platform crash through recursive thread creation, uncontrolled CPU or memory resource consumption, as well as hanging thread, decompression bomb and excessive size of manifest size.

Relative to default OSGi Platforms, Hardened OSGi Recommendations bring an important step toward building secure systems. When used together with Java Permissions, Hardened Felix enables the development of platforms that have a high Protection Rate. Hardened Felix with permissions has a Protection Rate of 66 %. Implementing Hardened OSGi Recommendations within the Equinox implementation would provide a Protection Rate of 74 %.

Hardened OSGi Recommendation fulfill their objective, namely hardening the vulnerabilities of the OSGi Platform that are introduced by OSGi itself. However, major Java vulnerabilities are still open, such as the 'recursive thread creation' which causes a platform crash, or uncontrolled CPU or memory resource consumption. Hardened OSGi Recommendations are thus necessary to develop secure platforms, but it is far from being sufficient since the underlying virtual machine has severe security drawbacks when used for multi-application systems.

4.3. Evaluation of Hardened Felix: Performances

The benefits of *Hardened OSGi* recommendations must also be evaluated according to the performance impact they have on the system. Conditions of the experiments are first given, and results are provided and commented.

The tests are performed on a i686 architecture, with a Pentium M processor of 1,86 GHz. Since the modifications concern for the vast majority of them the installation phase of the bundles, only this phase is considered. The performances during the execution are likely to be impacted by the Java Permissions, and not by the Hardened OSGi architecture, and are thus out of the scope of this study. Five profiles are considered: a void OSGi Platform (started

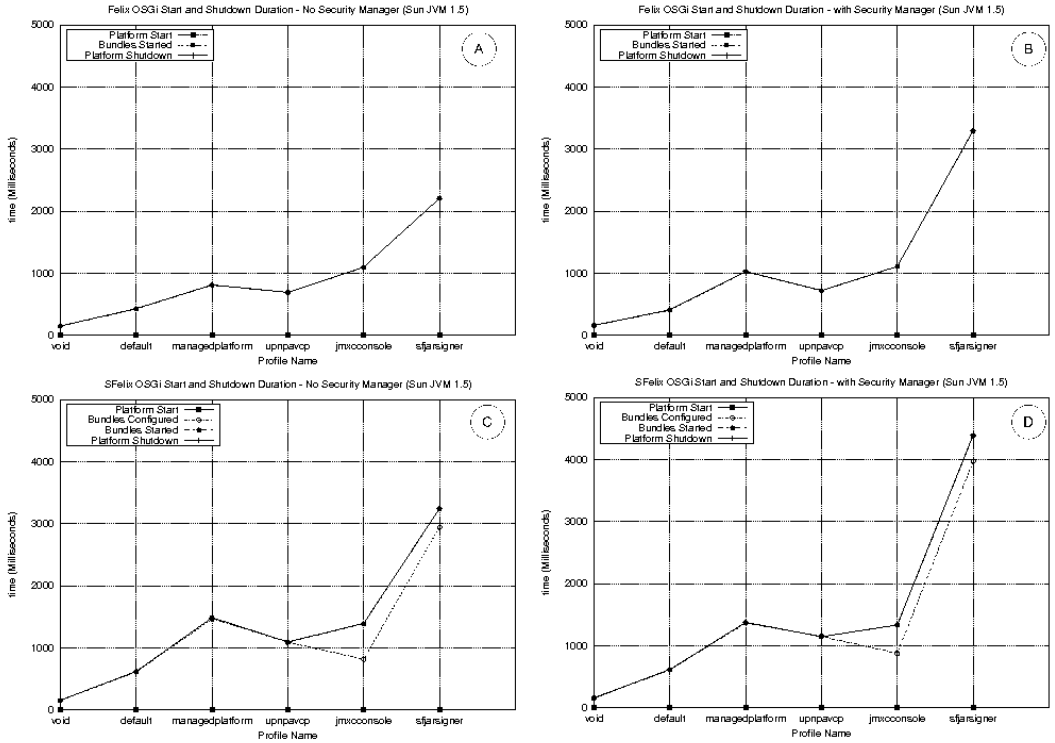


Figure 8. Compared performances of Felix and Hardened Felix (noted SFelix), without and with Java Permissions

without any bundles), a default Felix Platform (with the OSGi shell for local management, and the OBR client for discovering and downloading new bundles), the 'JMX manageable Platform' profile (that support remote management through a JMX-based Console), the 'UPnP Control Point' profile (that can play the role of UPnP gateway in intelligent Homes), and the applicative profiles 'JMX Console' and 'SF-Jarsigner'[‡‡] (which supports the digital signature and publication of OSGi bundles onto OBR repositories).

Tests are therefore performed for the following configurations as shown in the Figure 8: OSGi instance without (on the top) and with (on the bottom) integration of the Hardened OSGi Recommendations, and without (on the left side) and with (on the right side) the Java Permissions.

[‡‡]http://sf-jarsigner.gforge.inria.fr/

Performances of 'Hardened Felix' (C, D) are weaker than those of Felix (A, B), except in the case of the void profile: this is due to the verification of the validity of the bundle digital signature, which is relatively heavyweight - and not available in most OSGi implementations, which are highly security unaware. Starting an application without Java Permissions (A, C) is quicker that starting it with these Permissions (B, D). This difference is important in particular in the case of execution of complex programs, such as the starting process of the 'SF-Jarsigner' tool.

The other main modification in the platform behavior that is introduced by the Hardened OSGi Recommendations' is the fact that the bundle activator is launched in a separate thread. Consequently, two phases must be considered: the time point where the platform is available for receiving new commands (dotted lines), and the time point where the launching of the whole bundles is effective (plain line). The second phase is negligible in the case of basic activation processes (such as in the void, default, JMX manageable Platform, or UPnP Control Point). The two-phase launch behavior of Hardened OSGi improves the reactivity of the system, and is especially appreciated for embedded or handheld devices [22], in the case where the activation is a complex process, for instance when Graphical User Interfaces (GUIs) are started. This modifications thus also imply a better usability (and not only dependability) of the OSGi platforms.

As shown by the test results, other Recommendations only have a negligible impact on the performance of OSGi Platforms, since they are made of lightweight controls or actions. The performance measures also show the overhead that is implied by Java Permissions.

The Hardened OSGi Recommendations enable to build OSGi Platforms that are robust against some ill-coded bundles, or bundles that exploit some of the vulnerabilities that are described in this study. The overhead of securing an OSGi Platform can not be considered as negligible. Main performance loss are implied by Bundle Digital Signature and Java Permissions. The proposed Hardened Architecture in itself has no noticeable performance drawbacks. On the contrary, by using separate threads for executing applications, it provides better usability in addition to the dependability performance.

A secure flavour of the OSGi Platform is required to prevent the exploitation of the following attack vector, that is specific to Extensible Component Platforms: dynamic and transparent extension of the system at runtime. Recommendations for hardened OSGi Platforms are given which provide a great improvement in the security status of these platforms. These recommendations are validated through qualitative evaluation with the Protection Rate Metric and through performance evaluation.

Hardened OSGi recommendations tackle almost all the vulnerabilities that are due to the OSGi specifications themselves. However, Java specific vulnerabilities are not addressed, and build an obvious requirement for future work.

5. Conclusions and perspectives

The contribution of this study is threefold. First, the security benchmarking of several open source OSGi Platforms is performed, that enables us to provide recommendations for building

secure OSGi-based Systems. Secondly, tools are defined to support the security engineering process for such systems: a specific Vulnerability Pattern is introduced, as well as the Protection Rate metric which expresses the efficiency of security mechanisms. Thirdly, a Hardened implementation of Apache Felix is provided. It supports the validation of our recommendations and tools.

This study also put to light a strong requirement for further research efforts in order to build secure OSGi-based systems. In particular, we identify Five Security Challenges that are currently not prevented through any available security mechanism for the OSGi Platform:

1. **Infinite loop/hanging Thread in method call**: any bundle can contain non-returning calls,
2. **Memory load injection**: any bundle can consume memory,
3. **Decompression Bomb**: though it is identified by anti-virus software, this vulnerability is not prevented in OSGi-based systems,
4. **Exponential thread number**: any bundle can cause the Platform to crash due to memory exhaustion,
5. **Service Short circuit**: the numerous and optional service management mechanisms in OSGi do not support security enforcement so far.

The most urgent of these challenges is to patch the *Exponential thread number* attack, with implies the most serious damages to the platform.

An important conclusion of this work is that more than the half of the vulnerabilities in an OSGi Platform are not bound with OSGi itself, but to the underlying Java Virtual Machine. This means that all Java-based (and probably most VM-based systems) have a high probability to be weak relative to the presented attacks.

With this study, we intend to increase the knowledge of the security implication of Java-based Extensible Component Platforms, in particular the OSGi Platform. We also intend to solve most of OSGi-specific security issues by patching identified vulnerabilities. An important research effort is still required to achieve the long term objective that is made technically possible by the OSGi Platform: running secure multi-applications systems with mutually untrusted bundles.

ACKNOWLEDGEMENTS

This work is partially founded by MUSE IST FP6 Project #026442

REFERENCES

1. T. F. Arnold, The Concept of Coverage and Its Effect on the Reliability Model of a Repairable System IEEE Trans. Comput., IEEE Computer Society, 1973, 22, 251-254.
2. Arvidsson, J.; Cormack, A.; Demchenko, Y. & Meijer, J. (2001),'TERENA's Incident Object Description and Exchange Format Requirements', IETF RFC 3067.
3. Aslam, T. (1995),'A Taxonomy of Security Faults in the Unix Operating System', Master's thesis, Purdue University.

4. Bieber, G. & Carpenter, J. (2001), 'Introduction to Service-Oriented Programming (Rev 2.1)', OpenWings Whitepaper.
5. Black, P.E. (2005),Software Assurance Metrics And Tool Evaluation, in 'Proceedings of the 2005 International Conf. on Software Engineering Research and Practice (SERP'05)'.
6. Bloch, J. (2001), Effective Java Programming Language Guide, Addison-Wesley Professional.
7. Bloch, J. & Gafter, N.Addison-Wesley, ed. (2005), Java Puzzlers - Traps, Pitfalls and Corner Cases, Pearson Education.
8. Cotroneo, D., Orlando, S. & Russo, S. (2006), Failures classification and analysis of the Java Virtual Machine, in '26th IEEE International Conference on Distributed Computing Systems (ICDCS'06)'.
9. Czajkowski, G., Dayns, L. & Titzer, B. (2003), A Multi-User Virtual Machine, in 'Usenix', pp. 85–98.
10. Dean, Felten & Wallach (1996), Java Security: From HotJava to Netscape and Beyond, in 'SP '96: Proceedings of the 1996 IEEE Symposium on Security and Privacy', IEEE Computer Society, Washington, DC, USA, pp. 190.
11. Drossopoulou, S. & Eisenbach, S. (1997), Java is type safe - Probably, in 'ECOOP'97 - Object-Oriented Programming', Springer Berlin / Heidelberg, , pp. 389-418.
12. EISPP Consortium (2003),'EISPP Common Advisory Format Description', EISPP Whitepaper EISPP-D3-001-TR.
13. Gong, L., Ellison, G. & Dadgeforde, M. (2003), Inside Java 2 Platform Security - Architecture, API Design, and Implementation, Second Edition., Addison-Wesley.
14. Gong, L., Mueller, M., Prafullchandra, H. & Schemers, R. (1997), Going Beyond the Sandbox: An Overview of the New Security Architecture in the Java Development Kit 1.2, in 'Proceedings of the USENIX Symposium on Internet Technologies and Systems'.
15. Howard, M., J. Pincus & Wing, J. (2005), Computer Security in the 21st Century, Springer, chapter Measuring Relative Attack Surfaces, pp. 109-137.
16. Howes, T. (1997),'The String Representation of LDAP Search Filters', IETF RFC, Network Working Group, Request for Comments: 2254.
17. Jensen, T., Metayer, D.L. & Thorn, T. (1998), Security and Dynamic Class Loading in Java: A Formalisation, in 'IEEE International Conference on Computer Languages', pp. 4–15.
18. JSR 118 Expert Group (2002),'MIDP 2.0', Sun Specification.
19. Lindholm, T. & Yellin, F. (1999), The Java(TM) Virtual Machine Specification (2nd Edition), Prentice Hall PTR.
20. Lindqvist, U. & Jonsson, E. (1997), How to Systematically Classify Computer Security Intrusions, in 'IEEE Symposium on Security and Privacy', pp. 154-163.
21. Martin, R.A. (2005), 'Transformational Vulnerability Management Through Standards', CrossTalk, The Journal of Defense Software Engineering, 12-15.
22. Mikkonen, T. (2007), Programming mobile devices : an introduction for practitioners, Wiley.
23. Mowbray, T.J. & Malveau, R.C. (1997), Corba Design Patterns, John Wiley & Sons.
24. Neumann, P.G. & Parker., D.B. (1989), A summary of computer misuse techniques, in 'Proceedings of the 12th National Computer Security Conference', pp. 3961107.
25. OSGI Alliance (2007), 'OSGi Service Platform, Core Specification Release 4.1', Draft.
26. OSGi Alliance (2001), 'RFC 18: Security Architecture Specification', OSGi Alliance Request for Comment.
27. OSGi Alliance & Hall, R.S. (2006), 'Bundle Repository', OSGi Alliance RFC 112.
28. Parrend, P. & Frenot, S. (2007), 'Java Components Vulnerabilities - An Experimental Classification Targeted at the OSGi Platform'(RR-6231), Technical report, INRIA, 84 p..
29. Parrend, P. & Frenot, S. (2007), Supporting the Secure Deployment of OSGi Bundles, in 'First IEEE WoWMoM Workshop on Adaptive and DependAble Mission- and bUsiness-critical mobile Systems, Helsinki, Finland'.
30. Saraswat, V. (1997), 'Java is not type-safe', AT&T Research Whitepaper.
31. Schumacher, M. (2003), Security Engineering with Patterns, Springer Verlag.
32. Sun Microsystems, I. (2003),'JAR File Specification', Sun Java Specifications.
33. Sun Microsystems, I. (2002),'Java Security Architecture Specification', Sun Whitepaper

Table VII. Example of the Vulnerability Pattern:
Management Utility Freezing - Infinite Loop

```
<?xml version="1.0" encoding="UTF-8"?>
<vp:vulnerability
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xmlns:vp="http://sfelix.gforge.inria.fr/VulnerabilityPattern">
  <vp:reference>
    <vp:vulnerability_name>Management Utility Freezing \\
       - Infinite Loop</vp:vulnerability_name>
    <vp:extends>Infinite Loop in Method Call</vp:extends>
    <vp:identifier>
      <vp:catalog_id>mb</vp:catalog_id>
      <vp:src_ref>osgi</vp:src_ref>
      <vp:id>4</vp:id>
    </vp:identifier>
    <vp:origin>Ares research project ‘malicious-bundle’</vp:origin>
    <vp:location>Bundle Activator</vp:location>
    <vp:sources>
      <vp:source>
        <vp:entity>OSGi Platform - Life-Cycle Layer</vp:entity>
        <vp:related_flaw_or_function>
          <vp:flaw>No safe Bundle Start</vp:flaw>
        </vp:related_flaw_or_function>
      </vp:source>
    </vp:sources>
    <vp:target>OSGi Element - Platform Management Utility</vp:target>
    <vp:consequence_type>Performance Breakdown</vp:consequence_type>
    <vp:consequence_type>Unavailability</vp:consequence_type>
    <vp:introduction_time>Development</vp:introduction_time>
    <vp:exploit_time>Bundle Start</vp:exploit_time>
  </vp:reference>
  <vp:description>
    <vp:description_text>an infinite loop is executed in the Bundle \\
Activator</vp:description_text>
    <vp:preconditions>-</vp:preconditions>
    <vp:attack_process>an infinite loop is executed in the Bundle \\
Activator</vp:attack_process>
    <vp:consequence_description>block the OSGi Management entity (the \\
        felix, equinox or knopflerfish shell; when launched in the KF \\
        graphical interface, the shell remain available but the GUI is \\
        frozen). Because of the infinite loop, most CPU resource is consumed
</vp:consequence_description>
```

Table VIII. Example of the Vulnerability Pattern:
Management Utility Freezing - Infinite Loop (II)

```
    <vp:see_also>CPU Load Injection, Infinite Loop in Method Call, Non \\
Perturbating Infinite Loop, Hanging Thread</vp:see_also>
  </vp:description>
  <vp:implementation>
    <vp:code_reference>fr.inria.ares.infiniteloopinmethodcall-0.1.jar \\
</vp:code_reference>
    <vp:osgi_profile>J2SE-1.5</vp:osgi_profile>
    <vp:date>2006-08-24</vp:date>
    <vp:test_coverage>10%</vp:test_coverage>
<!-- for ease of use, type 'osgi_platform list' is defined as a XSD list -->
<vp:tested_on>
      <vp:vulnerable_platform>Felix</vp:vulnerable_platform>
      <vp:vulnerable_platform>Equinox</vp:vulnerable_platform>
      <vp:vulnerable_platform>Knopflerfish</vp:vulnerable_platform>
      <vp:vulnerable_platform>Concierge</vp:vulnerable_platform>
      <vp:robust_platform>SFelix</vp:robust_platform>
    </vp:tested_on>
  </vp:implementation>
  <vp:protection>
    <vp:existing_mechanisms>-</vp:existing_mechanisms>
    <vp:enforcement_point>-</vp:enforcement_point>
    <vp:potential_mechanisms>
      <vp:potential_mechanism>
        <vp:potential_mechanism_name>Code static Analysis \\
</vp:potential_mechanism_name>
      </vp:potential_mechanism>
      <vp:potential_mechanism>
        <vp:potential_mechanism_name>OSGi Platform Modification - Bundle \\
Startup Process</vp:potential_mechanism_name>
        <vp:potential_mechanism_descr>launch the bundle activator in a \\
separate thread to prevent startup hanging</vp:potential_mechanism_descr>
      </vp:potential_mechanism>
</vp:potential_mechanisms>
    <vp:attack_prevention>-</vp:attack_prevention>
    <vp:reaction>-</vp:reaction>
  </vp:protection>
</vp:vulnerability>
```

2.4 Project VOSGi[Royon et al. 2006]

Yvan Royon, Stephane Frenot and Frederic Le Mouel

Virtualization of Service Gateways in Multi-provider Environments
In CBSE 2006, 29/6-01/7, Västeras near Stockholm, Sweden

Virtualization of Service Gateways in Multi-provider Environments

Yvan Royon, Stéphane Frénot, and Frédéric Le Mouël

INRIA Ares - CITI Lab - INSA Lyon
Bat. Leonard de Vinci, 69621 Villeurbanne cedex, France
{yvan.royon, stephane.frenot, frederic.le-mouel}@insa-lyon.fr

Abstract. Today we see more and more services, such as entertainment or home automation, being brought to connected homes. These services are published and operated by a variety of service providers. Currently, each provider sells his own box, providing both connectivity and a closed service environment. The open service paradigm aims at mixing all services within the same box, thus opening the service delivery chain for home users. However, open service gateways still lack important mechanisms. Multiple service providers can access and use the same gateway concurrently. We must define what this use is, *i.e.* we must define a notion of *user*. Also, service providers should not interfere with each other on the gateway, except if explicitly required. In other words, we must isolate services from different providers, while still permitting on-demand collaboration. By combining all these mechanisms, we are defining a multi-user, multi-service execution environment, which we call a virtualized service gateway. We implement part of these features using OSGi technology.[1]

Keywords: Virtual gateway, multi-user, service-oriented programming.

1 Introduction

During the last years, high speed connectivity to the home has evolved at a very fast pace. Yesterday, home network access consisted in bringing IP connectivity to the home. The services made available were common application-level programs, such as web or e-mail clients. Today, the operators are moving to integrating value-added services in their offer, mainly multicast TV and Voice over IP. These network-enabled services are provided by the same connectivity box, or by a dedicated set-top box. It is foreseen that in the next few years, both the number and the quality of available services will increase drastically: home appliances, entertainment, health care... However, these services would be developed, maintained and supervised by other parties, for instance respectively whitegoods manufacturers, the gaming industry, and hospitals. Until today, the entire service chain and the delivery infrastructure, including the home gateway, are under the control of a single operator. Emerging standards push towards open solutions that enable both integration and segmentation of various markets such as connectivity, entertainment, security, or home automation. This

[1] This work was partially supported by the IST-6thFP-507295 MUSE Project

approach implies that, on the single access point that connects the home network to the internet (*i.e.* the home gateway), many service vendors are each able to deploy and manage several services.
Current and ongoing service platform efforts enable multi-party service provisioning, but they still lack strong concepts and mechanisms to completely support it. In particular, no isolation of services that belong to different parties has been defined.

We propose an isolation of services, depending upon which party, or *user*, deploys, manages and owns them. Consequently, we also define the notion of user in this context. By isolation, we mean that a service provider should only be able to "see" her own services on the common service platform. We represent this as a *virtual service gateway*. Each service provider owns and manages his own virtual gateway and the services it runs. All virtual gateways are run and managed by a unique *core service gateway*, typically hosted inside the home gateway (physical box), and operated by the home gateway provider.

The paper is structured as follows. Section 2 describes ongoing works on multi-application Java environments. Section 3 defines the notion of virtual service gateway in this context. Section 4 explains how we implement these concepts on top of OSGi. Finally, section 5 discusses and concludes the article.

2 Multi-application Java Environments

This paper focuses on Java-based environments. Java applications are architecture- and OS-agnostic, which is an interesting feature when using very diverse hardware platforms (*e.g.* home gateways). Also, Java Virtual Machines are available and already deployed on lots of architectures, which range from enterprise servers through PCs to mobile phones. Reusing the existing software base is a key to technology acceptance. The main alternative to Java is Microsoft's .NET. Our main concern with .NET is the lack of unloading facilities for assemblies. This would cause the environment to grow huge if applications were often updated. Therefore, in this section, we examine solutions to make the JVM a multi-application environment, which is essential for an open service gateway.

2.1 Current Java Environments

A standard Java Virtual Machine is a multi-thread but mono-application environment (figure 1). In order to run two applications, two JVMs are launched. In this case, the applications run independently, *i.e.* if they need to collaborate, they must access the operating system's communication facilities (*e.g.* TCP/IP network stack, file system...). We can see that the problems with this solution are both the overhead from running two JVMs, and the inefficiency of communications, even though there are proposals to limit these [1].

There are two kinds of responses to these insufficiencies: bringing multi-application capabilities to the JVM, or using an overlay on top of the JVM, *e.g.* a J2EE or similar application server.

2.2 Multi-application Java Environments

• Sun's Multi-tasking Virtual Machine [2] (figure 2), for instance, runs several Java applications, called *isolates*, in the same Java environment. Isolates share class representation, so that only static fields are duplicated. Applications are instrumented using a resource management interface, for heap memory management in particular.

• Rival proposals are Java operating systems [3] [4]. These mix the JVM with the operating system layer, and often come with multi-process capabilities.

• A last option is to add multi-application-like functionalities using an overlay on top of the JVM (figure 3). The overlay is the single application that runs in the JVM, but it allows several pseudo-applications to run concurrently on top of it.

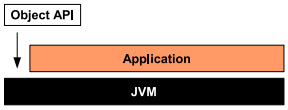


Fig. 1. Mono-application

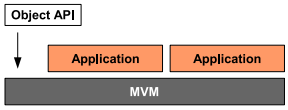


Fig. 2. Multi-task

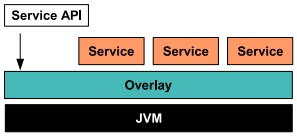


Fig. 3. Multi-service

2.3 Isolation Terminology

The term "isolation" may imply several kinds of mechanisms. We attempt here to basically classify them.

• The first family of mechanisms is namespace isolation. A namespace is a context for identifiers, and, in our case, for applications or services. Applications in different namespaces cannot "see" each other: this is an access right enforcement. With Java technology, this namespace isolation may be achieved through the use of classloaders, or more advanced loading facilities such as the Module Loader [5] or MJ [6].

• The second family of isolation mechanisms concerns low-level resources. In a resource-isolated environment, applications are supposed to be protected from one another. For instance, schedulers provided by operating systems allocate CPU slots for applications according to their priority. Recent Linux kernels also endorse out-of-memory kills, *i.e.* if a process endangers the whole system using too much memory, it gets killed. Such memory protection can be qualified as a reactive mechanism, versus proactive mechanisms. An example of a proactive mechanism would be Xen's hypervisor [7].

There are two ways to combine namespace isolation and resource isolation. The first one is to complete namespace isolation with reactive resource isolation, for instance by checking specific constraints such as CPU usage [8]. The second one is to build a combination of proactive, strong resource isolation and namespace isolation through the use of different virtualization techniques. Proposals such as Xen [7] or Denali [9] run multiple lightweight or full-featured operating systems on the same machine. Other attempts, such as Java *isolates*, provide an isolation API for Java applications.

2.4 Our Goals

The advantages of a modified runtime are performance (communications, memory sharing) and resource isolation (see below). Inversely, the advantages of overlays are their usability on any standard JVM, and their ease of development through their level of abstraction. Table 1 summarizes this comparison.

	Namespace isolation	Resource isolation	Performance optimizations	Uses a standard JVM	Easiness of integration[2]
Modified JRE	yes	yes	yes	no	intrusive
Overlay	no	no	no	yes	effortless

Table 1. Summary of Multi-application Java Environments

Our goal in this paper is to define a multi-user, service-oriented Java environment, without modifying the standard Java Runtime Environment. This implies that we use an overlay. Our contribution is divided in two steps: first we add namespace isolation to the overlay solution, then, we add a definition of users.

3 Towards a Multi-user, Service-oriented Environment

In this section, we evoke the notion of service-orientation and its benefits. We then detail the terms multi-user, through the definition of *core* and *virtual* service gateways. We explain on one hand how they should be isolated, and on the other hand when and how they should be able to cooperate.

3.1 Service-oriented Programming

For clarification, the term "service" in this paper refers to Service-Oriented Programming (SOP). SOP is based on Object-Oriented Programming, which relies on ideas such as encapsulation, inheritance and polymorphism. SOP additionally states that elements (*e.g.* components) collaborate through behaviors. These behaviors, also called services or interfaces, allow to separate what must be done (the contract) from how it is done (its implementation) [10].

Some service-oriented solutions, such as Web Services, deal with the interoperability in communications, and are referred to as Service-Oriented Architectures. By contrast, SOP environments such as the OSGi Service Platform [11] and Openwings are centered on the execution environment, which is the focus of this paper.

3.2 Namespace Isolation

Core and Virtual Gateways. A core service gateway is a software element, managed by an operator. It makes resources available in order to run services.

[2] Easiness of integration means the easiness of developing the environment itself, using it, and developing applications for it.

Such resources, physically supplied by the underlying hardware (*i.e.* the home gateway), include CPU cycles, memory, hard disk storage, network bandwidth, and optionally standard services (*e.g.* logging, HTTP connectivity). The gateway operator grants service providers access to these resources. This access is symbolized by a virtual service gateway, and provides a namespace isolation. Figure 4 illustrates this architecture.

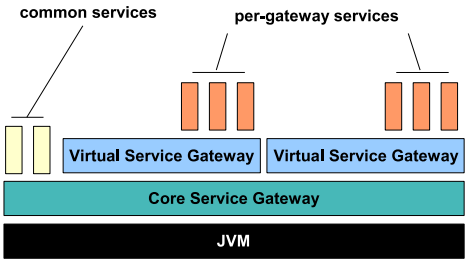


Fig. 4. Multi-service, namespace-isolated environment

Service cooperation. In an isolated, multi-application environment, applications are cloistered by default. However, they still should be able to cooperate on demand. In resource isolated environments, they cooperate through data communications. They either use standard OS facilities (*e.g.* sockets, IPCs, filesystem), or dedicated facilities (*e.g.* Links in the Java *isolates* API). By contrast, in open, non-isolated service environments, applications pass references on services (or interfaces). In an open, multi-service, multi-user, namespace-isolated environment, this still must be possible if explicitly permitted. The framework is then responsible for passing these references.

3.3 Multi-user Java Environment

The gateway operator, through the core service gateway, acts much like a Unix root user. He allows users (service providers) to launch their shell or execution environment (their virtual service gateway). The core gateway also runs services accessible to all users. However, contrary to Unix root users, the core gateway does not have access to service gateways' data, files, *etc*, since these would belong to different, potentially competing companies. Figure 5 represents the architecture with participating users.

The root user, *i.e.* the gateway operator, is responsible for the management of the virtual gateways it runs. This management layer is structured around 4 activities. Lifecycle management provides a mean to start and stop virtual gateways. Performance management provides information about the current status of a gateway (a virtual gateway or the core gateway itself). Security management positions credentials and make security challenges with core and virtual gateways. Finally, Accounting and Logging brings information about service usage for each gateway.

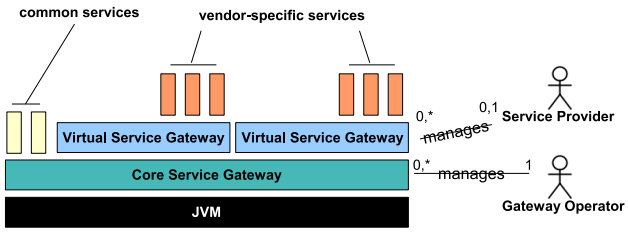


Fig. 5. Multi-service, multi-user residential gateway

Users, or service providers, access their virtual gateways through a remote monitoring interface. According to the business model described in section 1, each service provider is responsible for the bundles she deploys. This means that service providers are encouraged to supervise their own services on their clients' service gateways. Also, some business services may inherently need remote monitoring: health care, home automation...

4 Implementation

4.1 OSGi and Virtualized OSGi

The service-oriented overlay we use for our prototype implementation is the OSGi Service Platform [11]. The OSGi specification defines a service-oriented API (figure 3), deployment units (components called bundles), and a container (the service platform) which guarantees dependencies resolution for hosted components.

In order to create virtual gateways, we launch several OSGi gateway instances from within a core OSGi instance (figure 4). The core gateway also instruments and manages virtual gateways. This solution allows us to create a straightforward matching with the concepts of root user and users detailed above.

4.2 Service Isolation

The advantages of running several service gateway instances inside a single core gateway instance are quite immediate. Each service gateway can only access OSGi bundles and services it directly hosts. The core gateway itself does not see the hosted virtual gateways, but only a management agent that allows their life cycle management. This is a straightforward way to enforce namespace isolation.

Each service provider sees his own virtual gateway as if it was a standard OSGi service platform. Therefore, at deployment time, standard OSGi deployment schemes come in action. At runtime, namespace isolation is provided through a hierarchy of classloaders. Each deployed component (*i.e.* OSGi bundle) comes with its own classloader, which delegates to its service gateway's classloader [12]. This way, by default, services from different providers (*i.e.* running in different virtual service gateways) are in different namespaces.

4.3 Service Cooperation

The core service gateway can provide services to its virtual service gateways. Currently, a static list of shared services and implementations is passed from the core gateway to virtual gateways. Each virtual gateway then needs to internally publish these shared services, so that its own services may access them. A more dynamic solution is planned, but not yet implemented.

4.4 Performance

We chose to run several OSGi Framework instances inside a core Framework instance. The main drawback to this approach is that it induces a resource consumption overhead. We ran a first set of performance tests on a standard Pentium IV PC, using a Gentoo Linux operating system, and Sun's JDK 1.5 with standard parameters (*e.g.* initial memory allocation pool). Our measures compare a vanilla Oscar 1.0.5 gateway with a core gateway that runs six virtual gateways, each launching a standard set of bundles. After 24 hours, we observe an overall 33% increase in memory use within the JVM's pre-allocated memory pool. This corresponds to a 2.9 MB consumption overhead. More thorough benchmarks are planned and under progress.

4.5 Available Code

We provide an OSGi bundle[2] that allows to start and stop virtual OSGi instances. The project, called `vosgi` for Virtual OSGi, has been tested on patched versions of both ObjectWeb's Oscar[3] and Apache's Felix[4] open source implementations of the OSGi Service Platform specifications. The management activities expressed in section 3 are enabled through a JMX architecture [13] called `mosgi` (for Managed OSGi).

5 Discussion and Conclusion

In this paper, we have proposed a first step toward a multi-user, service-oriented execution environment. It can target the same markets as OSGi service platforms (mobile phones, vehicles, home gateways...), while improving isolation and management.
Since we use a standard Java virtual machine as the lower layer, our best option for service provider separation is to provide a namespace isolation. If we want to go further into resource isolation, we need a JVM and an operating system that provides such a functionality. For instance, we could provide a multi-user environment on top of real-time virtual machines. But these are not available on

[2] Available at `http://ares.inria.fr/~sfrenot/devel/` under the CeCILL open source licence.
[3] `http://oscar.objectweb.org/`
[4] `http://incubator.apache.org/felix/`

a large scale yet, and they are not aimed at this "in-the-middle" market (neither embedded nor high-end PCs).

The proposed multi-user environment currently has two main alternatives. The first one is the multi-tasking virtual machine, and the second one is the use of "standard" operating systems (*e.g.* Linux, Windows).
From the MVM point of view, Sun's team works on mapping OS-level users rights within the JVM [14]. This project is aimed at big server systems that host many users and applications (SPARC/Solaris). Inversely, our approach is focused on embedded systems, using standard JREs. Also, we believe that cooperation through service sharing, or interface sharing, is a better abstraction for developers than data sharing (`Link` objects between isolates).
Compared to standard operating systems, we assume that a service-oriented approach is a real improvement over "classical" C programming. We argue that both layers (Java and service orientation) are beneficial to service development for the targeted market. It enables many advantages (code structure, code hotplugging...) with only few drawbacks (mainly startup time).

References

1. Czajkowski, G., Daynès, L., Nystrom, N.: Code sharing among virtual machines. ECOOP (2002)
2. Czajkowski, G., Daynès, L.: Multitasking without comprimise: a virtual machine evolution. In: OOPSLA, New York, NY, USA, ACM Press (2001) 125–138
3. Golm, M., Felser, M., Wawersich, C., Kleinoeder, J.: The JX operating system. In: USENIX. (2002) 45–58
4. Prangsma, E.: JNode. (http://jnode.sourceforge.net)
5. Hall, R.S.: A Policy-Driven Class Loader to Support Deployment in Extensible Frameworks. In: Component Deployment, Edinburgh, UK. (2004) LNCS 3083, pp. 81 - 96.
6. J. Corwin, D.F. Bacon, D.G., Murthy, C.: MJ: a Rational Module System for Java and its Applications. In: (OOPLSA. (2003) pp. 241-254.
7. Barham, P., Dragovic, B., Fraser, K., Hand, S., Harris, T., Ho, A., Neugebauer, R., Pratt, I., Warfield, A.: Xen and the art of virtualization. In: SOSP. (2003)
8. Yamasaki, I.: Increasing robustness by code instrumenting: Monitoring and managing computer resource usage on OSGi frameworks. OSGi World Congress (2005)
9. Whitaker, A., Shaw, M., Gribble, S.: Denali: Lightweight virtual machines for distributed and networked applications (2002)
10. Bieber, G., Carpenter, J.: Introduction to service oriented programming. http://www.openwings.org (2001)
11. The OSGi Alliance: OSGi Service Platform. 4th edn. IOS Press (2005)
12. Liang, S., Bracha, G.: Dynamic class loading in the Java virtual machine. In: OOPSLA. (1998) pp. 36–44
13. Fleury, E., Frénot, S.: Building a JMX management interface inside OSGi. Technical report, Inria RR-5025 (2003)
14. Czajkowski, G., Daynès, L., Titzer, B.: A Multi-User Virtual Machine. In: Usenix. (2003) pp. 85–98

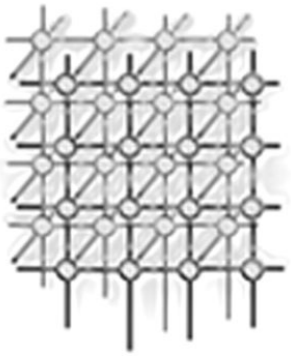
2.5 Project DARTS[Brebner et al. 2005]

Paul Brebner, Emmanuel Cecchet, Julie Marguerite, Petr Truma, Octavian Ciuh, Bruno Dufour, Lieven Eeckhout, Stéphane Frénot, Arvind S Krishna, John Murphy, Clark Verbrugge

A CPU Resource Consumption Prediction Mechanism for EJB deployment on a federation of servers

In Concurrency Computat. : Pract. Exper, 2005, vol 17, pp 1799–1805

CONCURRENCY AND COMPUTATION: PRACTICE AND EXPERIENCE
Concurrency Computat.: Pract. Exper. 2005; **17**:1799–1805
Published online 28 June 2005 in Wiley InterScience (www.interscience.wiley.com). DOI: 10.1002/cpe.918



Middleware benchmarking: approaches, results, experiences‡

Paul Brebner[1,*,†], Emmanuel Cecchet[2], Julie Marguerite[2], Petr Tůma[3], Octavian Ciuhandu[4], Bruno Dufour[5], Lieven Eeckhout[6], Stéphane Frénot[7], Arvind S. Krishna[8], John Murphy[4] and Clark Verbrugge[5]

[1]*CSIRO ICT Centre, G.P.O. Box 664, Canberra, ACT 2601, Australia*
[2]*INRIA Rhône-Alpes/ObjectWeb, 655 avenue de l'Europe, 38330 Montbonnot, St. Martin, France*
[3]*Department of Software Engineering, Charles University, Malostranské náměstí 25, Praha 1, Czech Republic*
[4]*Performance Engineering Laboratory, Dublin City University, Dublin 9, Ireland*
[5]*School of Computer Science, McGill University, Montréal, Québec, Canada H3A 2A7*
[6]*Department of Electronics and Information Systems (ELIS), Ghent University, St. Pietersnieuwstraat 41, B9000 Gent, Belgium*
[7]*INRIA Arès, INSA-CITI Bat. Léonard de Vinci, 69661 Villeurbanne Cedex, France*
[8]*Electrical Engineering and Computer Science, Vanderbilt University, Nashville, TN 37235, U.S.A.*

SUMMARY

The report summarizes the results of the Workshop on Middleware Benchmarking held during OOPSLA 2003. The goal of the workshop was to help advance the current practice of gathering performance characteristics of middleware implementations through benchmarking. The participants of the workshop have focused on identifying requirements of and obstacles to middleware benchmarking and forming a position on the related issues. Selected requirements and obstacles are presented, together with guidelines to adhere to when benchmarking, open issues of current practice, and perspectives on further research. Copyright © 2005 John Wiley & Sons, Ltd.

KEY WORDS: middleware benchmarking; middleware performance; middleware scalability; middleware evaluation; middleware benchmark design; benchmarking guidelines; benchmarking measurement; benchmarking metrics; OOPSLA 2003 workshop

*Correspondence to: Paul Brebner, CSIRO ICT Centre, G.P.O. Box 664, Canberra, ACT 2601, Australia.
†E-mail: paul.brebner@csiro.au

‡This report was produced from the OOPSLA 2003 Middleware Benchmarking Workshop, homepage http://nenya.ms.mff.cuni.cz/projects/corba/oopsla-workshop-03/.

Received 30 March 2004
Revised 18 May 2004
Accepted 10 June 2004

1. INTRODUCTION

For over a decade, middleware has been a well-established part of numerous applications. The spectrum of middleware implementations is diverse, ranging from communication-oriented middleware such as CORBA or SOAP libraries to component-oriented application servers such as the Enterprise JavaBeans (EJB) or CORBA Component Model (CCM) containers. Naturally, an important characteristic of these implementations is their performance under various conditions. Such a characteristic is used both by the middleware developers to improve and advertise their middleware and by the middleware users to decide on their middleware of choice.

The natural way to obtain the performance characteristics of a system is through benchmarking. However, although middleware benchmarking is a relatively frequent endeavor [1–11], the practice is rather fragmented. Middleware developers use proprietary testing suites with results that are rarely comparable across the spectrum of middleware implementations. Middleware users rely on simplistic testing suites whose results are often prone to misinterpretation when related to specific usage scenarios. Most standardization efforts in the area so far have also failed to bear fruit [12].

To remedy this situation, the participants of the Middleware Benchmarking Workshop held during OOPSLA 2003 shared their experience with designing, running and evaluating the existing benchmarks. As the next step, the participants identified some of the most significant obstacles encountered in the current practice and proposed approaches to tackle these obstacles. This report presents the results of the workshop in four sections, dedicated to the reasons for benchmarking, guidelines for benchmarking, open issues in benchmarking and perspectives on further research.

2. WHY BENCHMARKING?

An important statement emphasized repeatedly during the workshop debates was that the nature of a benchmark depends strongly on the intended use of the results. A benchmark that provides continuous data to an automated load-balancing system will be different from a benchmark that provides static data to a system developer. The somewhat obvious character of this statement is outweighed by the less obvious nature of its implications, which touch all aspects of benchmarking from the benchmark design through the measurement mechanisms to the processing of results. The following sections focus on the use of benchmarking for the design and the evaluation of middleware, and outline the implications of the considered use on the nature of the benchmark.

2.1. Benchmarking to design middleware

The first use of a benchmark that was considered important by the participants of the workshop was benchmarking to aid in the design of middleware. Essential to this use is benchmarking real applications to see how they stress the supporting system, collecting data to create models of supporting system usage typical for the selected application domains. Examples of such models are the workloads defined by the ECperf, RUBiS, SPEC jAppServer, SPEC JBB, Stock-Online and TPC-W benchmarks, all focusing on the online business domains. Additional models are needed for other domains.

Model-based techniques can be applied to visually represent techniques for defining entities and their interactions in the application domain using domain-specific building blocks. An example of

a model-based benchmarking tool is the CCMPerf benchmarking tool [10], which is a model-based benchmarking suite that allows developers and end-users of the CCM [13] middleware to evaluate the overhead that CCM implementations impose above and beyond the CORBA implementation, as well as to model interaction scenarios between CCM components using varied configuration options to capture software variability in higher-level models rather than in lower-level source code. The model-based techniques used in CCMPerf help domain experts visualize and analyze the results of performance experiments more effectively, since they focus on higher-level modeling abstractions, rather than wrestling with low-level source code.

During middleware design, benchmarking should be used to determine the optimal architecture for given constraints, such as memory capacity, processing power, or network throughput and latency. Important in this aspect is the task of validating models created during the design and, thus, predicting the real behavior of the architecture. Contemporary middleware is too complex to be understood purely through the analysis of its architecture, without benchmarking its real behavior.

Another important feature of the use of benchmarking to design middleware is the ability to capture and document consequences of using a specific architecture. An example of this approach is the work on design patterns for the server-side threading and dispatching models of TAO [14].

2.2. Benchmarking to evaluate middleware

The second use of a benchmark that was considered important by the participants of the workshop was benchmarking to evaluate middleware. This use of benchmarking includes comparing the performance of various middleware architectures, as well as comparing the performance of specific middleware configurations. In addition to the obvious use of the comparisons for selecting architectures and configurations, perspective uses include gathering data for system sizing and for determining system state and devising autonomic computing policies.

A benchmark needs to be able to evaluate the scalability of middleware. This implies benchmarking the behavior of the middleware and the supporting system when stretched to the limits of their scalability, as well as when used within the limits of their scalability.

3. BENCHMARKING GUIDELINES

The participants of the workshop also noted that many benchmarking projects tend to repeat common mistakes that devalue the results. Prominent among the common mistakes was an incorrect choice of metrics. Timestamps alone are not necessarily sufficient for understanding the results; annotating timestamps with system utilization and resource consumption data for the supporting system is useful.

A pitfall that can lead to useless results is using benchmarks that rely on an artificial workload such as the microbenchmarks of an isolated feature of the middleware. It is difficult to conclude anything about the behavior of real applications from the results of such benchmarks, and their use should therefore be limited and well justified.

The measurement technique in benchmarking is equally important. Particular performance measures or metrics may have a variety of intertwined effects, and a change or improvement in one measurement will usually affect many other quantities as well. A reduction in the number of method calls executed due to a method-inlining optimization, for instance, will increase code size, which may

alter cache performance. In order to make valid overall performance judgments, evaluations based on quantification must not only focus on providing a comprehensive representation of the benchmark that reveals important qualities, but also on understanding the inevitable dependencies between measurements.

Building a comprehensive view through measurement is itself challenging. A small, fixed suite of measurements is desirable since it makes comparisons simple and human comprehension feasible. Any reduced set of measures, however, may not give a true indication of benchmark activity and is certainly unlikely to capture all possible quantities of interest. A more reasonable approach is to permit a rough assessment through a few general measurements, but also calculate more specific measures in order to allow exploration of an apparent behavior through other perspectives and finer-grained detail. Evaluating a benchmark numerically is then a process of drilling down until one is satisfied that a clear understanding of the behavior has been determined [15].

Another pitfall is related to the interpretation of the results. The results must reflect the working of the measured feature and not interference from other features. That is especially true for warming up the system under test to a steady state to filter out interference such as cache priming, which is known to take minutes for simple benchmarks and can stretch to hours when complex mechanisms such as garbage collection or database access are involved. A similar concern applies to the interference from the workload injection limits with the system under test.

It may not be easy or straightforward to assess performance. Execution activity at various levels from the lowest to the highest will alter performance, and so must be understood in order to give an accurate evaluation. This is particularly true of low-level concerns; while the impact of higher-level algorithm design (and even of code generation choices) may be well understood by an application developer, low-level issues such as cache sizes, branch-predictor strategy and so on can have a significant impact on the actual performance cost of various benchmark actions. For instance, an interpretation that a benchmark is making a lot of method calls may not be indicative of excessive execution overhead if each call is correctly predicted by the processor. Therefore, a benchmark evaluation based on numerical data must be careful to account for all possible system features.

Particular care must be taken to understand, explore and document the impact of virtual machines for middleware that critically depends on them. This is an often poorly understood aspect of benchmarking Java middleware, which makes heavy demands on the Java Virtual Machine (JVM). The brand of JVM, number running per machine, settings, type of engine and type of garbage collector can have considerable (and not easily predicted) impacts on middleware performance and scalability, and hence benchmark results (see, for example, [16]).

As a general guideline, the participants of the workshop agreed that it is important to release exhaustive configuration information together with any results. In addition to the obvious need for reproducibility of the results, the information can be used for assessing the impact of configuration changes on the results.

Note that interpretation of numerical data will only be valid when the measurement is precisely defined. Many analyses fail to include enough information about how the measurement is actually calculated, making it not only difficult to reproduce and verify results, but also difficult to make comparative assessments. For example, lines of code, a traditional measurement of static application size, is easily perturbed by programming style and a variety of potential measurement possibilities (should it count blank lines, comment lines, etc.). Choices by different experimenters will often differ, and so exact metric definitions have an obvious necessity.

4. OPEN ISSUES

Paramount among the open issues was the question of the resources needed to conduct benchmarking, in terms of machinery, time to run the benchmarks and expertise needed to understand the complete setup of the benchmarks. The participants of the workshop agreed that the resource requirements are often prohibitive to conducting benchmarking, especially in a research environment with stringent requirements on results. An additional complication is that the lifetime of the expertise and the results is short, which increases the cost of benchmarking.

A benchmark should be indicative of the performance of a real application. This implies the need for real workloads, but such workloads tend to be unsuitable for benchmarking because of their size. Determining what substitute workloads are representative of real workloads is an important open issue.

During microprocessor design, computer architects face similar problems due to extremely long simulation times—it is not exceptional that simulating one second of a real system takes 24 hours. Several techniques have been proposed to address this issue in the context of uniprocessor performance modeling. *Sampling* [17–19] refers to selecting samples from a complete program run. Since the number of sampled instructions is smaller than the total number of instructions in a complete program run, significant simulation speedups can be obtained. Important issues that need to be dealt with are which samples to select and how to guarantee a correct hardware state at the beginning of each sample. A second technique is to use *reduced input sets* [20,21]. These reduced input sets are derived from longer reference inputs by a number of techniques: modifying inputs, truncating inputs, etc. The benefit of these reduced inputs is that the dynamic instruction count when simulating these inputs can be significantly smaller than for the reference inputs. A third approach is to apply *statistical simulation* [22–24]. The idea of statistical simulation is to define and collect a number of important program characteristics, to generate a synthetic workload from it that is several orders of magnitude smaller than a real application and, finally, to simulate this synthetic workload. If the synthetic workload captures the right characteristics, the behavior of the synthetic workload and the real application will be similar.

A key issue that needs to be dealt with for all of these techniques is to determine whether the substitute workload is representative of the real workloads. Previous work has shown that statistical data analysis techniques could be useful for this purpose [25]. We believe that adapting these methods to the context of middleware benchmarking could be extremely useful in speeding up measurement time while preserving the representativeness of the substitute workload. How to do this, however, remains an open issue.

With the middleware being just one part of a layered architecture, typically supported by the operating system or even another middleware, the results of middleware benchmarking depend not only on the middleware itself, but also on the supporting architecture. An open issue is how to abstract from the supporting architecture and characterize only the middleware layer, perhaps together with the interactions between layers.

A representative example of a layered architecture is the CCM [13], where the CCM applications sit on top of a CORBA ORB, which comprises three layers. The *host infrastructure layer* is the middleware layer that provides a uniform abstraction over the underlying operating system. The *distribution middleware layer* is the middleware layer such as CORBA that abstracts the data marshaling and connection management facilities. The *common middleware services layer* is the middleware layer that provides services. Benchmarks for CCM implementations are thus heavily

influenced by the underlying layers and it is necessary to characterize the influence of these layers in isolation.

An open issue of an entirely different nature is that of dealing with the publication of results for commercial products. Such results can often damage any of the concerned sides through real or perceived bias, or through simple misinterpretation.

5. PERSPECTIVES

With the numerous technical issues, it might be beneficial to keep a knowledge base of benchmarking expertise that would list such issues and thus help to keep research free of engineering mistakes. The question that remains unanswered is how to keep such a knowledge base up to date. Similar reasoning supports an open database of benchmarking results similar to the CORBA Benchmarking Project [9]. Such a database could help extract different information from the same results depending on the nature of the research that uses the results. The database would require solutions to the issues of anonymity, reliability and credibility of the results. The related technical issue of common data format would also need to be solved to facilitate sharing the results and the tools to process them.

Another beneficial step to the benchmarking efforts would be to get the middleware designers interested in supplying the recommended settings for specific applications and workloads. With representative workloads for selected application domains, this would help to avoid publishing results that are incorrect because of misconfiguration, especially for commercial products that often have complex or undocumented settings that influence benchmark results.

REFERENCES

1. Boszormenyi L, Wickener A, Wolf H. Performance evaluation of object oriented middleware—development of a benchmarking toolkit. *Proceedings of the 5th International Euro-Par Conference (EuroPar-99)* (*Lecture Notes in Computer Science*, vol. 1685). Springer Berlin, 1999.
2. Callison HR, Butler DG. *Real-time CORBA Trade Study*. Boeing, 2000.
3. Cecchet E, Chanda A, Elnikety S, Marguerite J, Zwaenepoel W. Performance comparison of middleware architectures for generating dynamic Web content. *Proceedings of the 4th ACM/IFIP/USENIX International Middleware Conference*, 2003. ACM Press: New York, 2003.
4. Cecchet E, Marguerite J, Zwaenepoel W. Performance and scalability of EJB applications. *Proceedings of the 2002 Object-Oriented Programming, Systems, Languages and Applications (OOPSLA-02). ACM SIGPLAN Notices* 2002; **37**(11).
5. Gokhale S, Schmidt DC. Measuring and optimizing CORBA latency and scalability over high-speed networks. *IEEE Transactions on Computers* 1998; **47**(4).
6. Juric MB, Rozman I. RMI, RMI-IIOP and IDL performance comparison. *Java Report* 2001; **6**(4).
7. Juric MB, Rozman I, Stevens AP, Hericko M, Nash S. Java 2 distributed object models performance analysis, comparison and optimization. *Proceedings of the 2000 International Conference on Parallel and Distributed Systems (ICPDAS-00)*. IEEE Computer Society Press: Los Alamitos, CA, 2000.
8. Tůma P, Buble A. Open CORBA benchmarking. *Proceedings of the 2001 International Symposium on Performance Evaluation of Computer and Telecommunication Systems (SPECTS-01)*. SCS, 2001.
9. Distributed Systems Research Group. CORBA Benchmarking Project, Charles University. http://nenya.ms.mff.cuni.cz.
10. Institute of Software Integrated Systems. CCMPerf: Model integrated test and benchmarking suite, Vanderbilt University. http://www.dre.vanderbilt.edu/∼arvindk/MIC/ccmperf.htm.
11. Gorton I, Liu A, Brebner P. Rigorous evaluation of COTS middleware technology. *IEEE Computer* 2003; (March):50–55.
12. Object Management Group. White paper on benchmarking. *OMG Document bench/99-12-01*, Object Management Group, 1999.

13. Wang N, Schmidt DC, O'Ryan C. An overview of the CORBA component model. *Component-Based Software Engineering*, Heineman G, Councill B (eds.). Addison-Wesley: Reading, MA, 2000.
14. Schmidt DC, O'Ryan C. Patterns and performance of distributed real-time and embedded publisher/subscriber architectures. *Journal of Systems and Software (Special Issue on Software Architecture Engineering Quality Attributes)* 2003; **66**(3):213–223.
15. Dufour B, Hendren L, Verbrugge C. Problems in objectively quantifying benchmarks using dynamic metrics. *Sable Technical Report 2003-6*, McGill University, 2003.
16. Brebner P. Is your AppServer being crippled by the JVM? *Proceedings of the 5th Annual Borland Conference Asia Pacific*, Sydney, 2002.
17. Conte TM, Hirsch MA, Menezes KN. Reducing state loss for effective trace sampling of superscalar processors. *Proceedings of the 1996 International Conference on Computer Design (ICCD-96)*. IEEE Computer Society Press: Los Alamitos, CA, 1996.
18. Sherwood T, Perelman E, Hamerly G, Calder B. Automatically characterizing large scale program behavior. *Proceedings of the 10th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS-X)*. ACM Press: New York, 2002.
19. Wunderlich RE, Wenisch TF, Falsafi B, Hoe JC. SMARTS: Accelerating microarchitecture simulation via rigorous statistical sampling. *Proceedings of the 30th Annual International Symposium on Computer Architecture (ISCA-30)*. ACM Press: New York, 2003.
20. Eeckhout L, Vandierendonck H, De Bosschere K. Designing computer architecture workloads. *IEEE Computer* 2003; **36**(2).
21. KleinOsowski J, Lilja DJ. MinneSPEC: A new SPEC benchmark workload for simulation-based computer architecture research. *Computer Architecture Letters* 2002; **1**.
22. Eeckhout L, Nussbaum S, Smith JE, De Bosschere K. Statistical simulation: Adding efficiency to the computer designer's toolbox. *IEEE Micro* 2003; **23**(5).
23. Nussbaum S, Smith JE. Modeling superscalar processors via statistical simulation. *Proceedings of the 2001 International Conference on Parallel Architectures and Compilation Techniques (PACT-2001)*. IEEE Computer Society Press: Los Alamitos, CA, 2001.
24. Oskin M, Chong FT, Farrens M. HLS: Combining statistical and symbolic simulation to guide microprocessor design. *Proceedings of the 27th Annual International Symposium on Computer Architecture (ISCA-27)*. ACM Press: New York, 2000.
25. Eeckhout L, Vandierendonck H, De Bosschere K. Quantifying the impact of input data sets on program behavior and its applications. *Journal of Instruction-Level Parallelism* 2003; **5**.
26. Center for Distributed Object Computing. The ACE ORB (TAO), Washington University. http://www.cs.wustl.edu/∼schmidt/TAO.html.

