



**HAL**  
open science

## Adaptation de services en environnement distribué

Erwan Daubert, Guillaume Gauvrit, Françoise André, Olivier Barais

► **To cite this version:**

Erwan Daubert, Guillaume Gauvrit, Françoise André, Olivier Barais. Adaptation de services en environnement distribué. RenPar'19, Sep 2009, Toulouse, France. inria-00411514

**HAL Id: inria-00411514**

**<https://inria.hal.science/inria-00411514v1>**

Submitted on 27 Sep 2009

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

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

# Adaptation de services en environnement distribué

Erwan Daubert, Guillaume Gauvrit, Françoise André, Olivier Barais

IRISA / INRIA  
Campus de Beaulieu  
35042 Rennes cedex – France  
{Erwan.Daubert, Guillaume.Gauvrit, Francoise.Andre, Olivier.Barais}@irisa.fr

---

## Résumé

L'industrie se tourne de plus en plus vers les architectures orientées services (*Service Oriented Architectures*, SOA) afin de construire des applications par composition de services existants et en cours de fonctionnement. La profusion des services et leur utilisation en environnement distribué dans un contexte de plus en plus dynamique engendre des besoins d'auto-adaptation qui n'ont pas encore été couverts par la communauté scientifique. Cet article présente un système d'adaptation générique pour les services et attaque la problématique de la migration de service, appliqué à l'architecture OSGi.

**Mots-clés :** auto-adaptation, services, distribution, migration, OSGi.

---

## 1. Introduction

L'utilisation d'architectures orientées services au sein d'entreprises est maintenant courante car elles offrent un paradigme de programmation adapté à la réalisation de systèmes d'information distribués. Héritières des architectures à composants, les architectures orientées services proposent la construction d'applications par assemblage de services en cours d'exécution pouvant être fournis par des tiers.

Certains de ces services, en particulier parmi ceux accessibles depuis l'extérieur de l'entreprise, peuvent être sujets à des contrats de qualité de service et à des sollicitations variables dans le temps. Aussi, afin de pouvoir satisfaire leurs contrats, ces services doivent être capables de s'adapter dynamiquement – c'est-à-dire au cours de l'exécution – à leurs utilisateurs et aux évolutions des caractéristiques de leurs environnements d'exécution, comme par exemple la charge de leur plate-forme hôte. Cette contrainte d'auto-adaptation n'est pas gérée par les plates-formes à services existantes, comme OSGi et SCA.

La plupart des travaux existants qui traitent de l'adaptation dynamique la considèrent dans le cadre des composants logiciels, comme Dynaco [1] ou SAFRAN [2]. Cependant les services offrent des possibilités d'adaptation supplémentaires qui ne sont donc pas traitées dans ces travaux, comme la migration de services. De plus, les résultats pratiques de ceux-ci ne peuvent être transposés directement sur les plates-formes à services car ils sont développés pour d'autres technologies.

Pour répondre à ce besoin d'un support pour l'auto-adaptation et la migration dynamiques de services, cet article présente un système générique d'adaptation de services et son implémentation avec OSGi.

## 2. Système d'adaptation et migration pour OSGi

Afin de permettre le développement de services auto-adaptables, nous proposons un système d'adaptation générique pour services, inspiré par des travaux effectués pour les composants [4, 1]. Ce système est conçu pour permettre tous types d'adaptations. Il sépare les fonctions d'adaptation, ce qui implique un code plus structuré et donc plus simple à concevoir et maintenir. Cela permet aussi une réutilisation de code, par exemple en partageant une politique d'adaptation entre différents services.

Le système d'adaptation utilise une structure à base de services et sépare les fonctions d'adaptation en services indépendants, ce qui permet de changer l'implémentation de tout ou partie du système

d'adaptation lors de son utilisation. Il respecte la décomposition de l'adaptation en phases successives d'observation, de décision, de planification et d'exécution proposée en [4].

Ce système a été implémenté avec la technologie OSGi. Une fonctionnalité de composition d'évènements a été intégrée au gestionnaire d'évènement, le service principal de la phase d'observation. Pour cela, nous avons utilisé le framework WildCat [3]. Le gestionnaire de service permet d'abstraire les évènements de leurs sources, de les centraliser et de les composer.

Nous avons en outre développé deux actions d'adaptation utiles dans le contexte de services s'exécutant sur des plateformes distribuées : la migration et la duplication de service. Ceci consiste à déplacer (respectivement dupliquer) un service d'une plate-forme d'exécution à une autre. L'intérêt est de limiter la charge sur l'une des machines ou encore de positionner, sur une machine moins chargée, un service consommateur en ressources. Cela peut aussi être utile afin de limiter l'utilisation des ressources réseau entre plusieurs services si l'on positionne ceux-ci sur la même machine.

Avec OSGi, plusieurs services peuvent être implémentés dans un même bundle, mais ne souhaitons pas nécessairement migrer l'ensemble des services du bundle avec celui-ci. Aussi nous utilisons un filtre pour sélectionner les services à démarrer. Pour gérer le problème de la sauvegarde de l'état du service, le modèle de développement proposé au développeur de services intègre le patron de conception *memento*. Nous avons défini le service *ServiceManagement* afin de prendre en charge la migration et duplication de service. Il permet aussi d'enregistrer ou retirer un service sur la plate-forme. Cependant ces différentes fonctionnalités ne peuvent être utilisées que sur des services se trouvant sur la même plate-forme OSGi que le *ServiceManagement* utilisé pour effectuer ces modifications. Ainsi il n'est pas possible de migrer un service d'une plate-forme A en utilisant le *ServiceManagement* présent sur une plate-forme B. De plus toutes ces opérations peuvent être effectuées uniquement sur un service ayant été enregistré sur la plate-forme par notre *ServiceManagement*.

Il est aussi nécessaire de gérer différentes contraintes propres à OSGi. Un bundle migré peut avoir des dépendances Java manquantes l'empêchant de s'initialiser. Pour résoudre ces dépendances, nous envisageons d'utiliser un *ClassLoader* réparti qui permettrait d'accéder à des bundles distants et ainsi limiter la charge sur la nouvelle plate-forme.

De plus, si des services requis par le bundles sont indisponibles sur la nouvelle plate-forme, il faut les rendre accessibles depuis celle-ci. D'une manière symétrique, il est nécessaire que le service migré reste accessible pour les utilisateurs sans interruption. Pour cela, il peut être nécessaire de créer un proxy sur l'ancienne plate-forme afin que les services qui se trouvent sur celle-ci puissent toujours l'utiliser.

### 3. Conclusion et perspectives

Pour répondre au manque de support pour l'auto-adaptation de services, nous avons présenté un système d'adaptation générique pour services en environnement distribué, avec une implémentation pour OSGi. Nous proposons aussi des outils pour OSGi qui permettent de migrer ou dupliquer un service au travers de plates-formes connectées entre elles. Par ailleurs, nous sommes en train d'implémenter des solutions aux problèmes identifiés en fin de section section 2.

Bien que nous puissions faire de la migration de service sur OSGi, il serait intéressant d'en faire de même entre différents types de plates-formes pour, par exemple, migrer un service OSGi sur une plate-forme SCA, ce qui faciliterait l'adaptation entre plates-formes hétérogènes. Nous comptons également approfondir l'étude des fonctions de décision et de planification en prenant en compte les possibilités offertes par les architectures orientées services. Nos travaux seront utilisés dans le cadre du réseau d'excellence Européen S-Cube (<http://s-cube-network.eu>).

### Bibliographie

1. Jérémy Buisson: *Adaptation dynamique de programmes et composants parallèles*. Thèse de doctorat, INSA de Rennes, 2006.
2. Pierre Charles DAVID: *Développement de composants Fractal adaptatifs : un langage dédié à l'aspect d'adaptation*. Thèse de doctorat, Université de Nantes UFR Sciences et Technique, 2005.
3. Pierre Charles David et Thomas Ledoux: *WildCAT : a generic framework for context-aware applications*. Dans *MPAC '05 : Proceedings of the 3rd international workshop on Middleware for pervasive and ad-hoc computing*, pages 1–7, New York, NY, USA, 2005. ACM, ISBN 1-59593-268-2.
4. J. Kephart et D. Chess: *The Vision of Autonomic Computing*. *IEEE Computer*, 36(1) :41–50, 2003.