Techniques d'intégration de services dans les environnements distribués

Noha Ibrahim, Frédéric Le Mouël, Stéphane Frénot

INRIA Ares, laboratoire CITI, INSA Lyon
Bâtiment Léonard de Vinci, 20 avenue Albert Einstein
69621 Villeurbanne Cedex
{noha.ibrahim, frederic.le-mouel, stephane.frenot}@insa-lyon.fr

RÉSUMÉ. *Les fournisseurs d'accès à Internet ont de plus en plus tendance à fournir des ensembles de services (WiFi, voie sur IP, télévision sur IP). Les applications s'exécutant dans ces environnements dynamiques doivent pouvoir intégrer les services fournis par différents fournisseurs et présents partout dans l'environnement. Dans cet article, Nous proposons un système d'intégration de services. Ce système fournit un cadre de conception (*framework*) composé d'une interface - `Integrable` - et d'un ensemble d'outils implémentant cette interface. Ces outils offrent différentes techniques d'intégration (composition locale ou distante, tissage locale ou distant, déploiement). Nous démontrons la faisabilité de notre système en implémentant un prototype basé sur la plateforme Java et la technologie OSGi*[1].

ABSTRACT. *The development of highly dynamic distributed environments modifies the runtime behavior of applications. Applications tend to use services available everywhere in the environment and would like to, whenever it is possible and/or needed, integrate services offered by the local environment. In this article, we propose a system for integration of services. Our system provides a framework including an interface - `Integrable` - and the tools implementing this interface. These tools offer different techniques of integration (local/remote composition, local/remote weaving, deployment by downloading/uploading). We demonstrated the feasibility of our system by implementing a prototype based on Java platform and OSGi technology.*

MOTS-CLÉS : *services, intégration, OSGi, systèmes distribués*

KEYWORDS: *services, integration, OSGi, distributed systems*

1. Ce travail est partiellement financé par le projet européen IST Amigo-Ambient Intelligence for the Networked Home Environment.

1re soumission à *5 ème atelier sur les Objets, Composants et Modèles dans l'Ingénierie des Systèmes d'Information*, le 21 avril 2006

1. Introduction

L'intégration de service se définit comme le problème de combiner différents services en un seul offrant une nouvelle fonctionnalité. Cette problématique suscite de plus en plus d'intérêt dans plusieurs communités et cela pour plusieurs raisons. Tout d'abord, le nombre de fournisseurs offrant des services nouveaux aux applications est en forte expansion (Kien *et al.,* 2005). Le développement de nombreux environnements distribués hautement dynamiques, comme les environnements pervasifs ou les grilles de calcul, modifie le comportement d'exécution des applications. L'intégration de service réduit le temps et l'effort des applications à chercher les services requis. En combinant les services, les applications utilisent ainsi de nouvelles fonctionnalités provenant de différents fournisseurs (Ponnekanti *et al.,* 2002).

L'intégration de service est ainsi un défi, à cause de la diversité des services fournis dans les environnements distribués. L'intégration demande souvent la maîtrise de plusieurs technologies et techniques. Notre objectif est de fournir un système d'intégration de services adapté aux environnements distribués. Ce système fournit un cadre de conception (*framework*) composé d'une d'interface - `Integrable` - et d'outils implémentant cette interface. Ces outils rendent possible l'application de différentes techniques d'intégration : composition locale ou distante, tissage locale ou distant, déploiement.

Actuellement, nombre de cadres de conception existent pour intégrer les services mais ces cadres sont destinés, très souvent, à un modèle précis de service et imposent une technique d'intégration. Notre cadre de conception est indépendant des outils qui l'implémentent, et une application peut très bien utiliser notre cadre avec ses propres outils.

Nous avons démontré la faisabilité de notre système en implémentant un prototype basé sur la plateforme Java et la technologie OSGi. Les outils que nous proposons sont spécifiques aux choix que nous avons fait. Le choix de Java a été motivé par sa portabilité et sa capacité à fournir une forte séparation entre les interfaces et leurs implémentations. Cette séparation reflète la séparation que nous avons faite entre notre cadre de conception et les outils l'implémentant. OSGi (OSGI Alliance, 2005) a été choisi pour sa capacité d'Inversion de Control (IOC) (Loritsch, 2001). OSGi simplifie le développement, le déploiement et la gestion des services en découplant les spécifications du service de son implémentation.

La force de notre système est dans son cadre de conception qui peut s'adapter aux plateformes existantes (.Net, OSGi, Fractal, J2EE, etc.). La limite de la version actuelle de notre système est dans ses outils ; les applications désirant utiliser ces outils doivent supporter la machine virtuelle Java et la plateforme OSGi.

Dans la section 2, nous présentons l'architecture de notre système. La section 3 détaille l'implémentation de notre prototype. La Section 4 examine les travaux liés à la problématique d'intégration de services. La Section 5 conclue et présente les perspectives de ce travail.

2. Architecture du système d'intégration de service

Cette section introduit l'architecture de notre cadre de conception (*framework*). Nous considérons que tous les services sont présents dans une couche middleware fournissant un environnement d'exécution. Nous y définissons un service spécial *IntegServ*. Ce service réside sur chaque machine désirant intégrer des services. L'intégration des services, qu'elle soit distante ou locale, est gérée par le service *IntegServ*. Nous allons tout d'abord définir notre modèle de service pour ensuite détailler l'architecture de notre système.

2.1. *Modèle de Service*

Notre modèle de service est constitué de trois parties (cf. figure 1) :

– Les interfaces : Une interface spécifie les méthodes qu'il est possible d'exécuter sur un service. Les interfaces d'un service sont publiques. Un service peut avoir deux types d'interfaces : les interfaces fonctionnelles définissant les fonctionnalités rendues pas le service et une interface d'intégration du service (l'interface `Integrable` définit la manière d'intégrer le service).

– Les liens établis : Un service fournit des fonctionnalités et a également besoin d'autres pour pouvoir s'exécuter. Les liens établis (ou *Bindings*) expriment les dépendances construites statiquement au développement ou dynamiquement à l'exécution.

– Les objets fonctionnels : Les objets réalisent les fonctionnalités attendues du service.

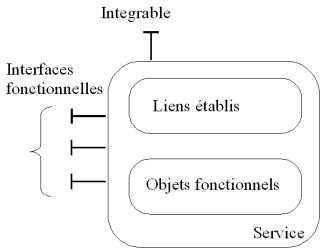

Figure 1. *Modèle de service*

Notre modèle de service est indépendant du choix de son implémentation et peut être appliqué aux EJBs (Monson-Haefel, 2000), composants CORBA (Zahavi, 1999), composants Fractal (Bruneton, 2004), bundles/services OSGi (OSGI alliance, 2004) et Services Web (.net) (Iverson, 2004). Les interfaces sont techniquement exprimées en utilisant un langage de programmation comme le langage Java (les interfaces java) ou en utilisant un langage de description d'interface (IDL, *Interface Description Language*), comme dans CORBA (Zahavi, 1999). Les liens établis peuvent être exprimés par un langage de description d'architecture (*Architecture Description Language*

ADL) (Rennie *et al.,* 2004) . L'implémentation des objets fonctionnels est dépendante du langage de programmation utilisé et résulte de l'instanciation des classes.

2.2. *Architecture du service* IntegServ

Plusieurs services s'exécutent sur différentes machines réparties dans un environnement distribué (cf. figure 2). Nous considérons deux types de services : les services basiques et les services intégrés. Un service qui veut pouvoir intégrer un autre service de l'environnement doit implémenter l'interface `Integrable`. Le service responsable de l'intégration dans l'environnement est le service *IntegServ*. Lorsqu'un appel à l'interface `Integrable` d'un service, est effectué, cet appel est redirigé vers un *IntegServ* local (étape 1 figure 2) ou, si aucun ne s'exécute sur la machine, vers un *IntegServ* distant. Un service de découverte prend en charge la tâche de chercher et trouver un *IntegServ* dans l'environnement (étape 2 figure 2). Notre système ne fournit pas le service de découverte mais en utilise des existants comme le service de découverte de Jini (Kumaran, 2002) ou UPnP (Microsoft Corporation, 2000).

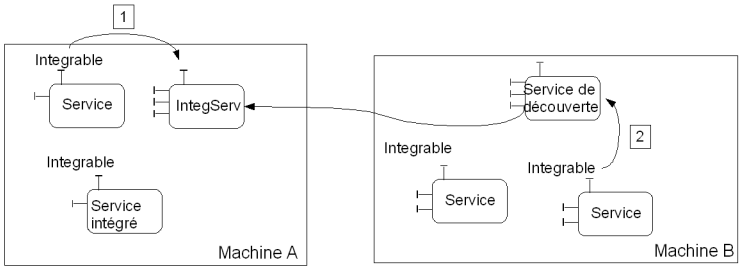

Figure 2. *Architecture du service* IntegServ

Le service *IntegServ* permet d'appliquer différentes méthodes d'intégration ciblant les parties internes du service (i.e. liens établis et objets fonctionnels) mais aussi les parties externes (i.e. interfaces). Ces différentes méthodes sont applicables l'une avant l'autre et peuvent être combinées. Elles ont toutes pour cible un ensemble de services. Le service *IntegServ* fournit trois techniques d'intégration :

– La composition de services est une technique d'intégration des services ciblant les interfaces. Le service *IntegServ* définit deux moyens pour connecter les services à travers leurs interfaces. La première est de republier toutes les interfaces des services à intégrer. Le service composé fournit ainsi toutes les interfaces des services participant à l'intégration (cf. figure 3). Le deuxième moyen est de fournir une seule interface qui redirige vers toutes les autres interfaces des services. Cette méthode est utilisée si les sorties d'une interface sont compatibles avec les entrées d'une autre, formant ainsi une chaîne d'interfaces compatibles (cf. figure 4). Bien sûre une combinaison de ses deux méthodes est possible.

– Le tissage des services permet d'entrelacer le code d'un objet d'un service, dans le code des objets d'un autre service. Dans cette technique d'intégration, seul les objets fonctionnels des services sont concernés (cf. figure 5). Le service tissé publie les même interfaces fonctionnelles qu'il possédait avant le tissage, mais la fonctionalité des méthodes de l'interface a bien changé.

– Le déploiement des services permet la distribution des services dans l'environnement en les téléchargeant sur différentes machines. Les liens établis du service sont alors concernés. Une fois un service déployé sur une machine, de nouveaux liens sont créés et les anciens détruits (cf. figure 6).

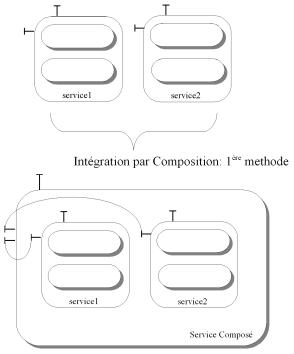

Figure 3. *Composition de services : première méthode*

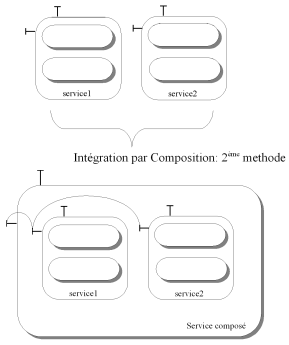

Figure 4. *Composition de services : deuxième méthode*

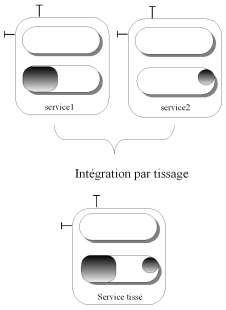

Figure 5. *Tissage de services*

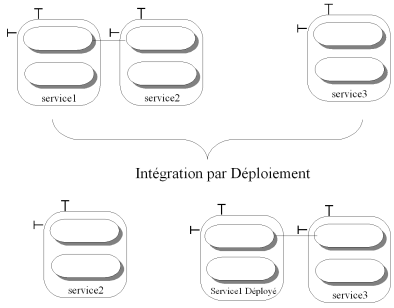

Figure 6. *Déploiement de services*

Rien n'empêche d'ajouter d'autres techniques d'intégration au service technique d'intégration. La seule condition est de respecter le modèle de service défini.

3. Premier prototype

Dans cette section, Nous présentons le développement de notre cadre de conception et l'implémentation de notre prototype sous Java et OSGi. Dans cet article, nous détaillons les trois techniques d'intégration proposées par notre service *IntegServ*.

3.1. *Cadre de conception*

L'interface `Integrable` fournit trois méthodes permettant la gestion de l'intégration de services : la méthode `integrate`, la méthode `getIntegratedServices` et la méthode `disIntegrate`. Si l'intégration/la désintégration n'est pas possible, une exception est levée (`IntegrationException`/`UnIntegrationException`). Le paquetage `IntegServ` (cf. figure 7) contient trois classes correspondant aux trois techniques d'intégration implantant l'interface `Integrable`. La classe `ByComposition` implémente la méthode `integrate` de l'interface `Integrable` en utilisant la technique de composition. La classe `ByWeaving` implémente la méthode `integrate` de l'interface `Integrable` en utilisant la technique de tissage. Enfin, la classe `ByDeployment` implémente la méthode `integrate` de l'interface `Integrable` en utilisant la technique de déploiement.

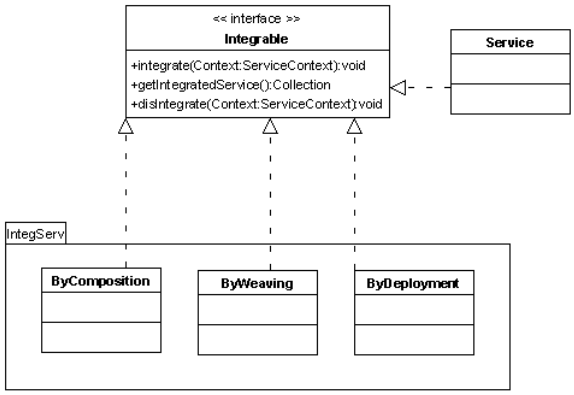

Figure 7. *Diagramme de classe du service IntegServ*

3.2. *Implémentation Java, OSGi*

OSGi (*Open Services Gateway initiative*) (OSGI Alliance, 2005) est une spécification standardisée pour le déploiement de services. Elle est initialement et principalement appliquée aux passerelles installées entre un réseau extérieur tel qu'Internet et un réseau local tel qu'un réseau domestique. Cette spécification permet de favoriser

le déploiement des services Java qui sont téléchargés dynamiquement sur les passerelles concernées et qui sont accessibles par tous les appareils du réseau interne qui sont connectés à cette passerelle. Nous avons implémenté notre cadre de conception sur la plateforme Oscar (Hall., 2005), une implémentation open-source d'OSGi. Nous avons enrichi Oscar avec notre service *IntegServ*. Nous appliquons notre modèle de service aux composants de déploiement d'OSGi (*bundles*). Les interfaces du modèle correspondent aux interfaces Java. Les objets fonctionnels sont les objets Java instanciés et exécutés avec un `Activator`. Les liens établis sont représentés par le fichier `manifest.mf`. Un service est fourni par un bundle. Un bundle peut également fournir plusieurs services.

3.2.1. *Scénario d'utilisation*

Dans cet article, nous avons choisi de détailler le code de nos trois techniques : déploiement des services par RMI, la composition des services (deuxième méthode) et le tissage des services. Ces différentes techniques peuvent être utilisées suivant le contexte et la nature des services. Si les machines sur lesquelles s'exécutent les services sont en mode connecté, une composition par RMI peut être appliquée. Si une des machines doit se déconnecter souvent du réseau, un re-déploiement des services peut être utile, suivi d'un tissage local. Si les machines ne sont pas tout le temps Nous définissons le scénario suivant illustrant nos différentes techniques. Soient deux services A et B s'exécutant sur deux machines différentes (cf. figure 8). La fonctionnalité du service B est rendu par un appel à la méthode B et le service A par un appel à la méthode A. Nous supposons que les sorties de la méthode A sont compatibles avec les paramètres d'entrées de la méthode B. Nous détaillerons les différentes techniques d'intégration en intégrant les deux services A et B.

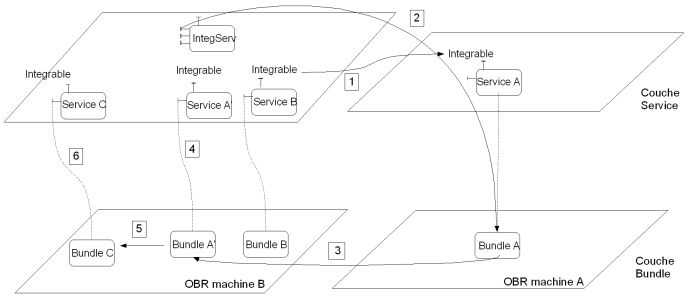

Figure 8. *Intégration des services A et B*

Le Service B décide d'intégrer le service A (étape 1 figure 8). Le service *IntegServ* présent sur la machine B, arrête le service A pour le télécharger sur la machine B (étape 2 & 3 figure 8), et l'exécuter (étape 4 figure 8). Une fois déployé, *IntegServ* le compose (ou le tisse) avec le service B. Un nouveau service C apparaît ainsi sur la machine B (étape 5 & 6 figure 8) proposant une combinaison des deux services A et B.

3.2.2. *L'appel d'intégration*

Le service B appelle la méthode `integrate` de l'interface `Integrable` avec comme paramètre le contexte distant contenant la machine A :

```
serviceB.integrate(context);
```

Listing 1 – L'appel à la méthode d'intégration

Le paramètre `context` de la méthode `integrate` est de type `ServiceContext` qui hérite de la classe `BundleContext` d'OSGi. Le `ServiceContext` contient une liste de services disponibles dans le contexte. Ce paramètre peut être de classe `ConcretContext` quand le contexte est local, c.a.d que le service à intégrer est sur la même machine. Il peut aussi être de classe `ConcretContextSerializable` quand le contexte est distant, c.a.d que le service à intégrer est sur une autre machine réseau.

3.2.3. *Technique de déploiement*

Les services sont techniquement fournis par des bundles. Sur chaque machine, nous utilisons un répertoire décentralisé et distribué d'Oscar (*Oscar Bundle Repository OBR*) où tous les bundles sont stockés. Le service *IntegServ* s'exécutant sur la machine B analyse l'emplacement du service A et décide de le télécharger de sa machine et de l'exécuter localement avant de l'intégrer avec le service B. L'implémentation de la méthode `integrate` (Listing 2) correspond à celle fournit par la classe `Bydeployment` (cf. figure 7). Le service A est alors arrêté et le bundle A est sérialisé et envoyé par flux (Listing 3).

```
ByDeployment mtc = new ByDeployment();
String location = "C:/OBR/bundleA.jar";
byte[] filedata = mtc.downloadBundle(location);
File file = new File(location);
BufferedOutputStream output = new
BufferedOutputStream(new FileOutputStream(file.getName()));
output.write(filedata,0,filedata.length);
output.flush();
output.close();
```

Listing 2 – Intégration par déploiement

```
File file = new File(fileName);
byte buffer[] = new byte[(int)file.length()];
BufferedInputStream input = new BufferedInputStream(new FileInputStream(fileName));
input.read(buffer,0,buffer.length);
input.close();
return(buffer);
```

Listing 3 – code de la méthode `downloadBundle`

3.2.4. *La Composition par redirection*

Une fois téléchargé, le bundle A est installé et exécuté. Le service A est ainsi disponible sur la machine B. Une composition locale (cf. figure 9) fourni par la classe

`ByComposition` (cf. figure 7) peut avoir lieu. Un nouveau bundle est ainsi créé offrant un service C accessible par la méthode `methodC()` et redirigeant vers les deux services A et B (s'ils sont disponibles). La méthode C lance la méthode A suivi de la méthode B . La classe `ByComposition` (figure 7) implémente ici la seconde méthode de composition (figure 4) parce qu'on suppose que les interfaces sont compatibles.

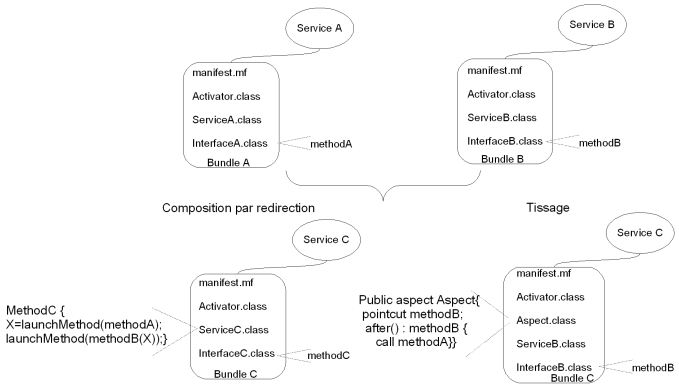

Figure 9. *Composition par redirection et tissage de services*

Le code de la méthode `launchMethod` est donné en Listing 4.

```
Object launchMethod(Object o, Object[] args,
          String nomMethode)     throws Exception{
  Class[] paramTypes = null;
  if(args != null){
    paramTypes = new Class[args.length];
    for(int i=0;i<args.length;++i)
      {paramTypes[i] = args[i].getClass();}
                }
  Method m = o.getClass().getMethod(nomMethode,paramTypes);
  return m.invoke(o,args);}
```

Listing 4 – code de la méthode `launchMethod`

3.2.5. *Technique de tissage*

Le service *IntegServ* peut appliquer une autre technique d'intégration pour intégrer les deux services A et B, le tissage (cf. figure 9). Nous distinguons deux différentes méthodes pour le faire (cf. figure 10). La première consiste à tisser dans le code du service B l'appel à la méthode A du service A. La deuxième consiste à tisser le code de la méthode A. Dans la deuxième technique, le service tissé peut ainsi exister même si le service A devient indisponible, alors que dans la première technique si le service A disparaît le service tissé ne peut plus fonctionner correctement.

Techniquement le service B est arrêté, *IntegServ* extrait les classes du bundle B. Pour la première technique, *IntegServ* tisse l'aspect et la classe Service B. Pour l'instant, nous avons utilisé AspectJ (Laddad, 2003) qui agit sur les classes. Pour cela,

nous lui fournissons un aspect en code source et après compilation nous exécutons le tissage avec le service B (cf. figure 9). Nous sommes en train de tester JAC (Pawlak *et al.,* 2004) pour son orientation plus objets que classes. Nous allons essayer de voir si nous pouvons utiliser la deuxième méthode de tissage (cf. figure 10) avec JAC. Le

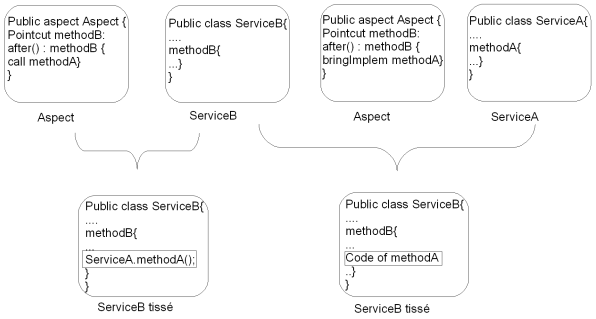

Figure 10. *Deux méthodes différentes de tissage*

code de l'aspect tissé sur le service B est donné dans le Listing 5.

```
public aspect Aspect {
 pointcut callMethodB() : call (public static * ServiceB.methodB(..));
 after(ServiceContext context) : callMethodB() {callMethodA(context);}
 void callMethodA(ServiceContext context) {
 try {
 ServiceReference[] refs = context.getServiceReferences(
           InterfaceA.class.getName(), "(service=serviceA)");
 if (refs != null) {
 InterfaceA comp = (InterfaceA ) context.getService(refs[0]);
 comp.methodA();}
  } catch (InvalidSyntaxException ex){}
 }
}
```

Listing 5 – Aspect code source compilé par AspectJ

Rien n'empêche d'enrichir notre prototype sous OSGi. La procédure est simple. Il suffit d'ajouter des implémentations de nouvelles techniques dans le paquetage *Integ-Serv*. Par exemple, ajouter une classe à ce paquetage qui implémente différemment la méthode `integrate` de l'interface `Integrable`. Rien n'empêche non plus d'utiliser d'autres outils basés sur une autre technologie qu'OSGi et qui respecte le modèle de service et le cadre de conception.

4. Etat de l'art

Trois domaines importants de la programmation par objets se penchent sur la problématique de l'intégration : la programmation orientée composants (*Component-Based Software Engineering CBSE*) (Heineman *et al.,* 2001), la programmation orientée aspects (*Aspect-Oriented Programming AOP*) (Kiczales *et al.,* 1997) et la

programmation orientée services (*Service-Oriented Programming SOP*) (Singh *et al.,* 2005). Les différents types de modèles à base de composants comme les EJBs (Monson-Haefel, 2000), le modèle à composant de CORBA (Zahavi, 1999), Fractal (Bruneton, 2004) permettent l'intégration de composants. L'intégration de composants dans ces différents modèles est souvent réduite au déploiement de composants et/ou à la paramétrisation de certains attributs de ces composants. La programmation orientée aspects permet d'implanter les préoccupations transverses (les aspects) indépendamment les unes des autres et de les combiner ultérieurement (le tissage) pour produire l'application finale. AspectJ (Laddad, 2003), FAC (Pessemier *et al.,* 2004) et JAC (Pawlak *et al.,* 2004) sont des modèles à base d'aspect appliquant le tissage d'aspect comme méthode d'intégration. Dans la terminologie de la programmation orientée service, l'intégration de service est souvent réduite à une composition de service. Actuellement, ces recherches visent à développer une architecture qui permet la composition de service en utilisant un raisonnement logique offert par les langages de description de service comme DAML (Sheshagiri *et al.,* 2003), UDDI et WSDL (Walsh, 2002). Les servicesWebs (Iverson, 2004) fournissent une solution pour la composition et l'assemblage des composants web en se basant sur des protocole XML.

5. Conclusion et travaux futurs

Dans cet article, nous avons présenté un système d'intégration de services. Notre système fournit un cadre de conception avec une d'interface - `Integrable` - et un ensemble d'outils implémentant cette interface. Notre cadre de conception est simple et s'applique à un modèle générique de service. Son utilisation est également simple, les services invoquent seulement la méthode `integrate` de l'interface `Integrable` sans avoir besoin de connaître les mécanismes impliqués. Pour démontrer la faisabilité de notre système, nous avons implanté l'ensemble d'outils correspondant à notre cadre de conception en utilisant les technologies Java et OSGi. Nous avons enrichi la plateforme OSGi avec notre service *IntegServ*. Dans cette article, Nous détaillons les différentes techniques de l'intégration : déploiement, la composition par redirection et le tissage. Cette implantation constitue également la limite de la version actuelle de notre système : les applications désirant utiliser ses outils doivent supporter la machine virtuelle de Java et la plateforme OSGi. Rien n'empêche toutefois l'application de notre modèle de service et notre cadre d'intégration à d'autres technologies. Dans le futur, nous visons l'ajout d'un protocole de négociation pour l'intégration. Pour se conformer aux attentes des services et ne pas forcément imposer l'intégration, les services pourront négocier et établir un contrat avant de s'intégrer.

6. Bibliographie

Bruneton E., *Developing with Fractal*, The ObjectWeb Consortium, France Telecom (R&D). March, 2004, version 1.0.3.

Hall. R. S., Oscar an OSGI framework implementation, Technical report, Objectweb organisation, 2005.

Heineman G. T., Councill W. T., *Component-Based Software Engineering : Putting the Pieces Together*, Addison-Wesley, June, 2001.

Iverson W., *Real Web services*, O'Reilly, October, 2004.

Kiczales G., Lamping J., Mendhekar A., Maeda C., Lopes C. V., Loingtier J.-M., Irwin J., « Aspect-Oriented Programming », *European Conference on Object-Oriented Programming (ECOOP)*, p. 220-242, June, 1997.

Kien T. N., Erradi A., Maheshwari P., « WSMB : a middleware for enhanced Web services interoperability », *Interop-ESA'05, First International Conference on Interoperability of Entreprise Software and Applications*, 2005. Geneva, Switzerland.

Kumaran S. I., *JINI Technology An Overview*, Prentice Hall PTR, 2002.

Laddad R., *AspectJ in Action : practical Aspect-Oriented Programing*, Manning publications, July, 2003.

Loritsch B., Developing With Apache Avalon, Technical report, Apache Software Foundation, 2001.

Microsoft Corporation, Understanding UPnP : A White Paper, Technical report, UPnP Forum, 2000.

Monson-Haefel R., *Entreprise JavaBeans*, O'Reilly & Associates, March, 2000.

OSGI alliance, About the OSGI Service platform, Technical report, OSGI alliance, July, 2004. revision 3.0.

OSGI Alliance, « OSGi Service Platform, Core Specification Release 4 », Draft, 07, 2005.

Pawlak R., Duchien L., Florin G., Legond-Aubry F., Seinturier L., Martelli L., « JAC : An Aspect-based Distributed Dynamic Framework », *Software Practise and Experience (SPE)*, vol. 34, n° 12, p. 1119-1148, 2004.

Pessemier N., Seinturier L., Duchien L., « Components, ADL and AOP : Towards a common approach. », *Workshop ECOOP Reflection, AOP and Meta-Data for software Evolution (RAM-SE04)*, June, 2004.

Ponnekanti S. R., Fox A., « SWORD : A Developer Toolkit for Web Service Composition », *11th World Wide Web Conference*, 2002. Honolulu, USA.

Rennie M. W., Misic V. B., Towards a Service-Based Architecture Description Language, Tr 04/08, University of Manituba, August, 2004.

Sheshagiri M., des Jardins M., Finin T., « A planner for composing services described in DAML-S », *International Conference on Automated Planning and Scheduling (ICAPS) 2003 Workshop on planning for web services*, July, 2003.

Singh M., Huhns M. N., *Service-Oriented Computing*, Wiley, December, 2005.

Walsh A. E., *UDDI, SOAP and WSDL : the Web Services specification Reference book*, Pearson Education, April, 2002.

Zahavi R., *Entreprise Application Integration with Corba Component and Web-Based solutions*, Johne Wiley & sons, November, 1999.