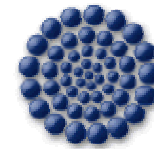




LSR-Adèle



SEP • CONACYT

Consejo Nacional de Ciencia y Tecnología

M É X I C O 

Vers un modèle à composants orienté services pour supporter la disponibilité dynamique

Humberto Cervantes

29 Mars 2004



Plan de la thèse

Problématique

Fondations

Modèle à composants orienté services

Réalisation

Évaluations

Conclusions et perspectives



Plan de la thèse

Problématique

- Disponibilité dynamique

Fondations

Modèle à composants orienté services

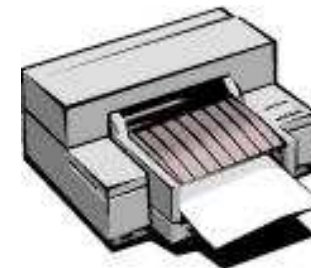
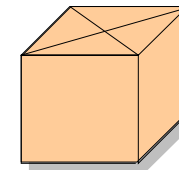
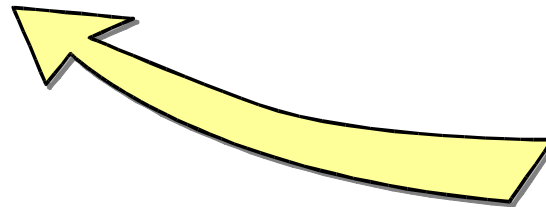
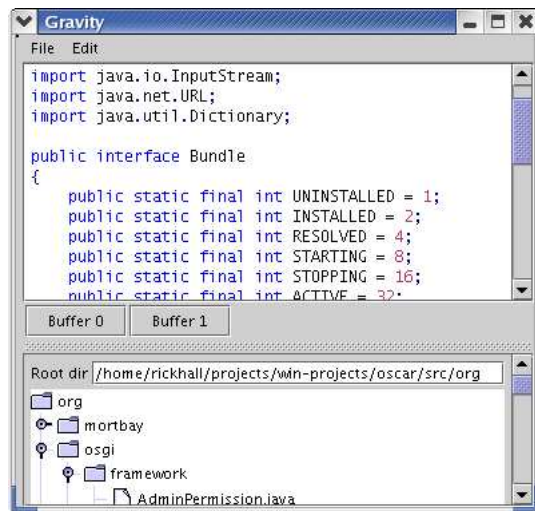
Réalisation

Évaluations

Conclusions et perspectives



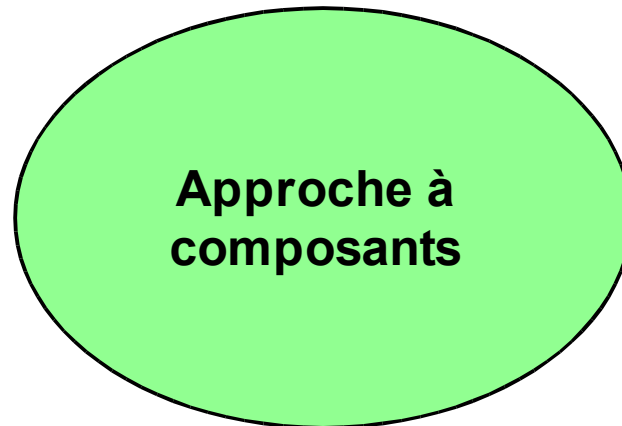
Disponibilité dynamique



Support *explicite* de la
disponibilité dynamique?



Approche

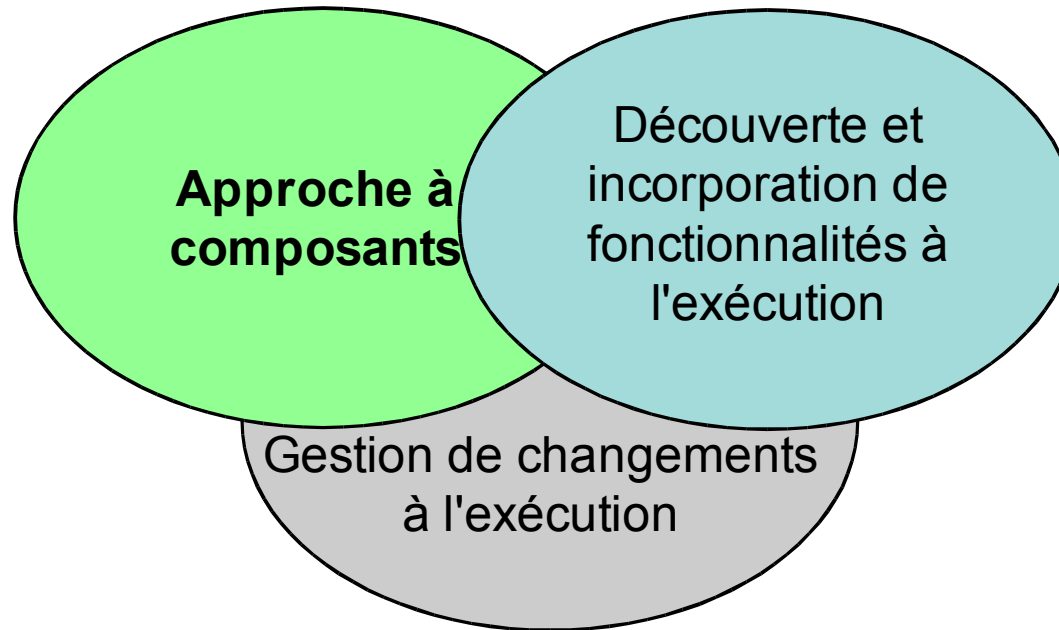


Composants: briques logicielles réutilisables

- Applications construites à partir d'assemblage
- Disponibilité dynamique n'est pas un concept de base

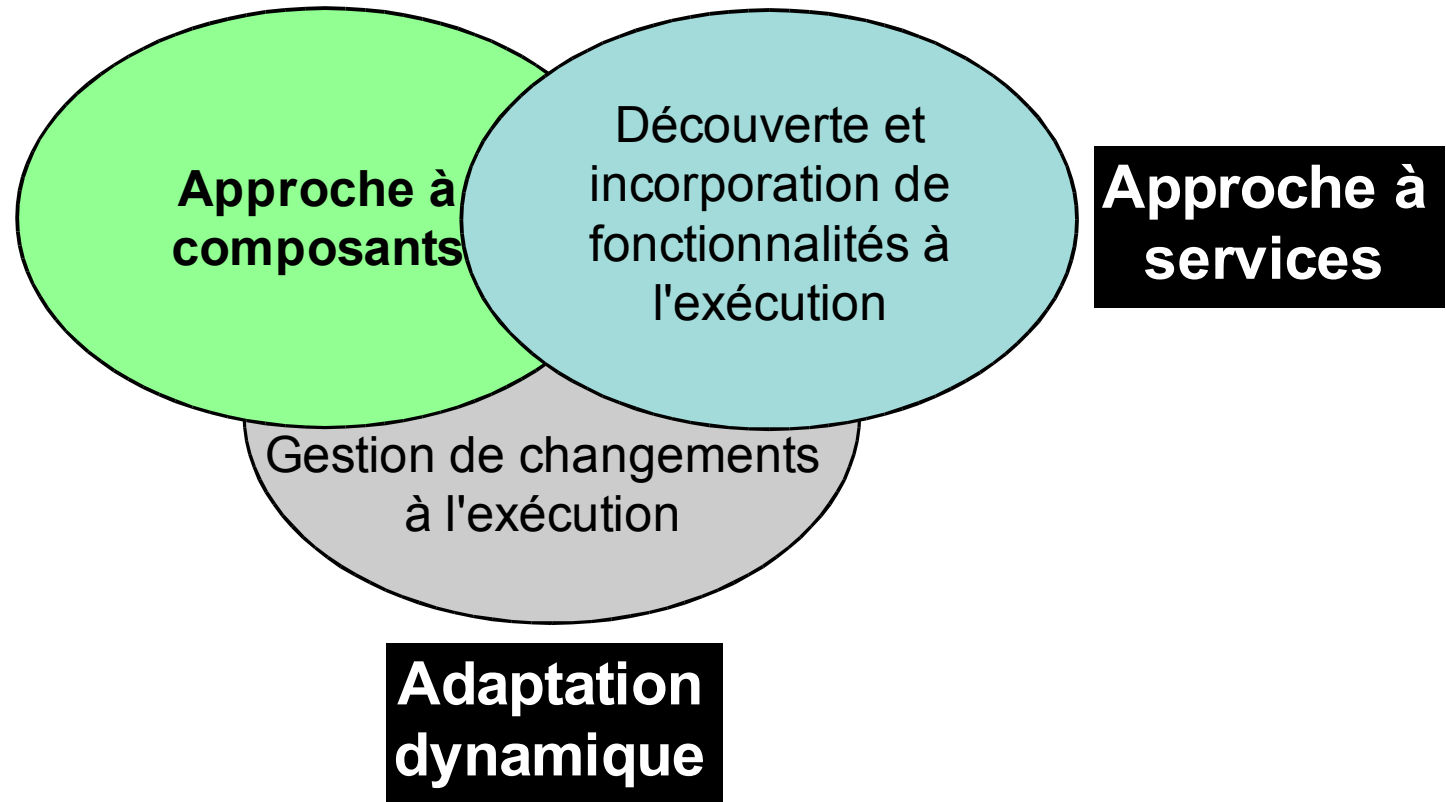


Approche



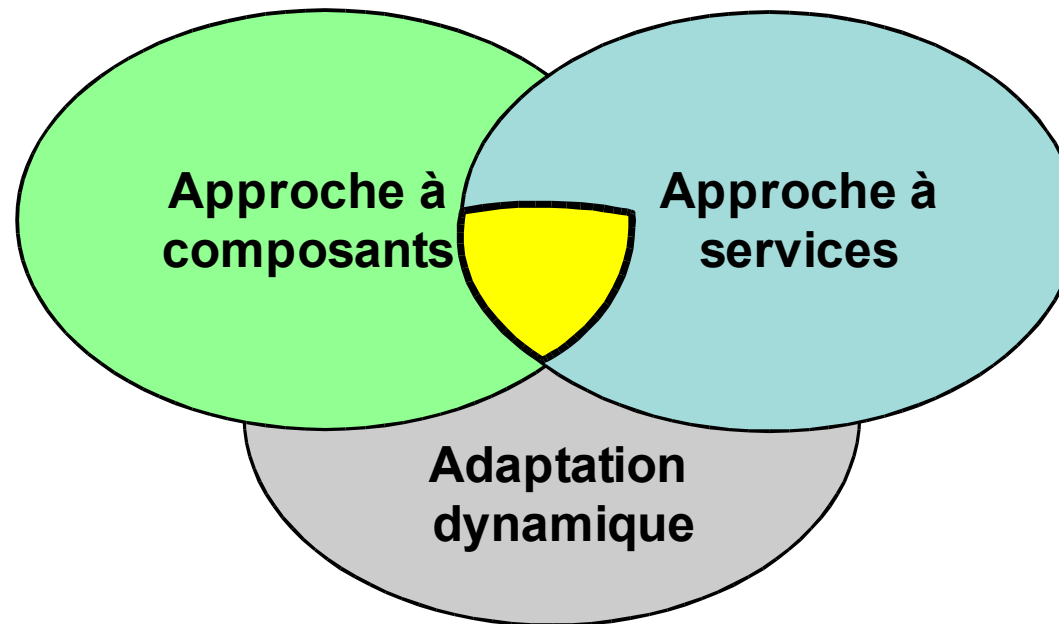


Approche





Approche

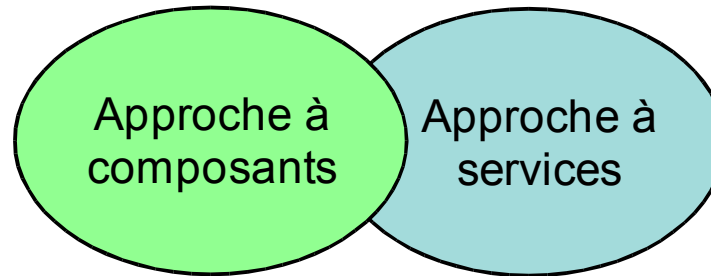


Modèle à composants orienté services

- Support *explicite* de la disponibilité dynamique dans un modèle à composants



Approches existantes



Combinaison concepts services et composants

- Avalon
- Web service components
- CCM et Traders

Se focalisent sur découverte en temps d'exécution

- Pas de gestion de la disponibilité dynamique

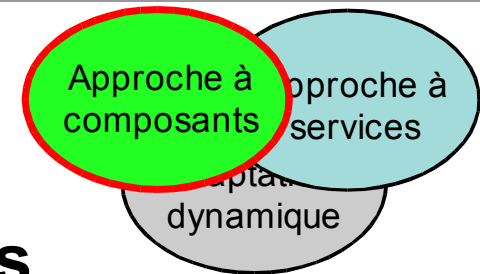


Plan de la thèse

Problématique

Fondations

- 🔍 ● Approche à composants
- Approche à services
- Adaptation dynamique



Modèle à composants orienté services

Réalisation

Évaluations

Conclusions et perspectives



Approche à composants

Définition de Szyperski

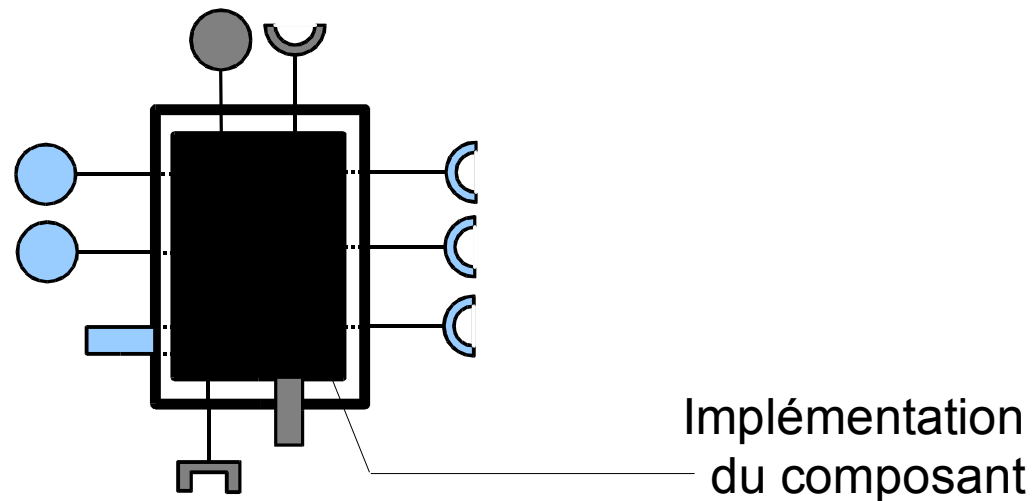
- unité binaire de composition
- interfaces spécifiées de façon contractuelle
- dépendances explicites
- déployé de façon indépendante
- sujet à composition par des tierces

Mais aussi...

- Paquetage
- Classe (composant)
- Instance



Éléments fournis et requis

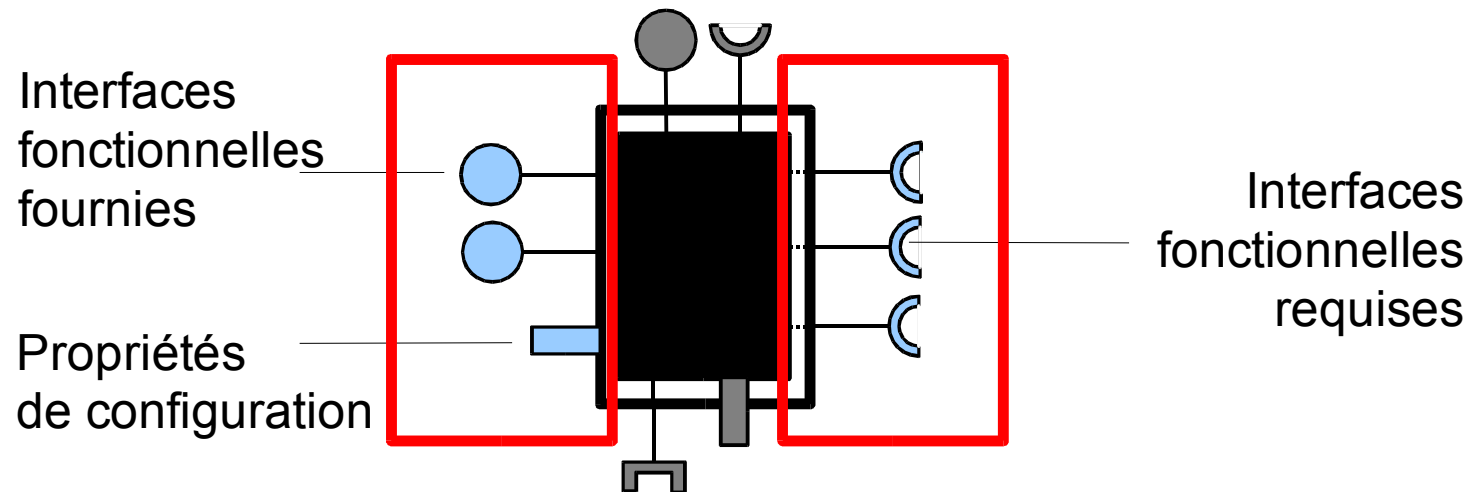


Composant définit explicitement ce qu'il fournit et ce qu'il requiert

- Assemblage
- Déploiement
- Gestion des instances à l'exécution



Éléments liés à l'assemblage

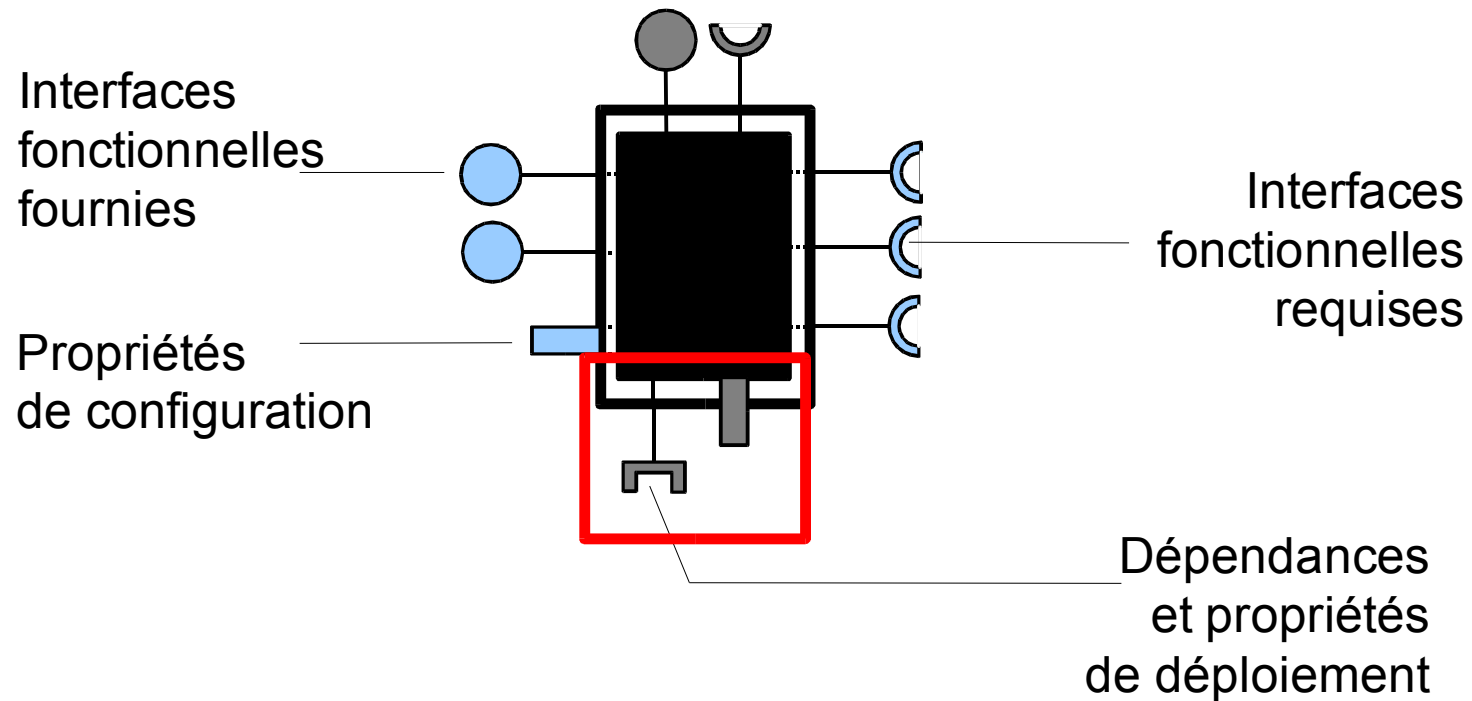


Représentent la *vue externe* du composant

- Ce que voient les clients
- Aspects fonctionnels
- Type de composant



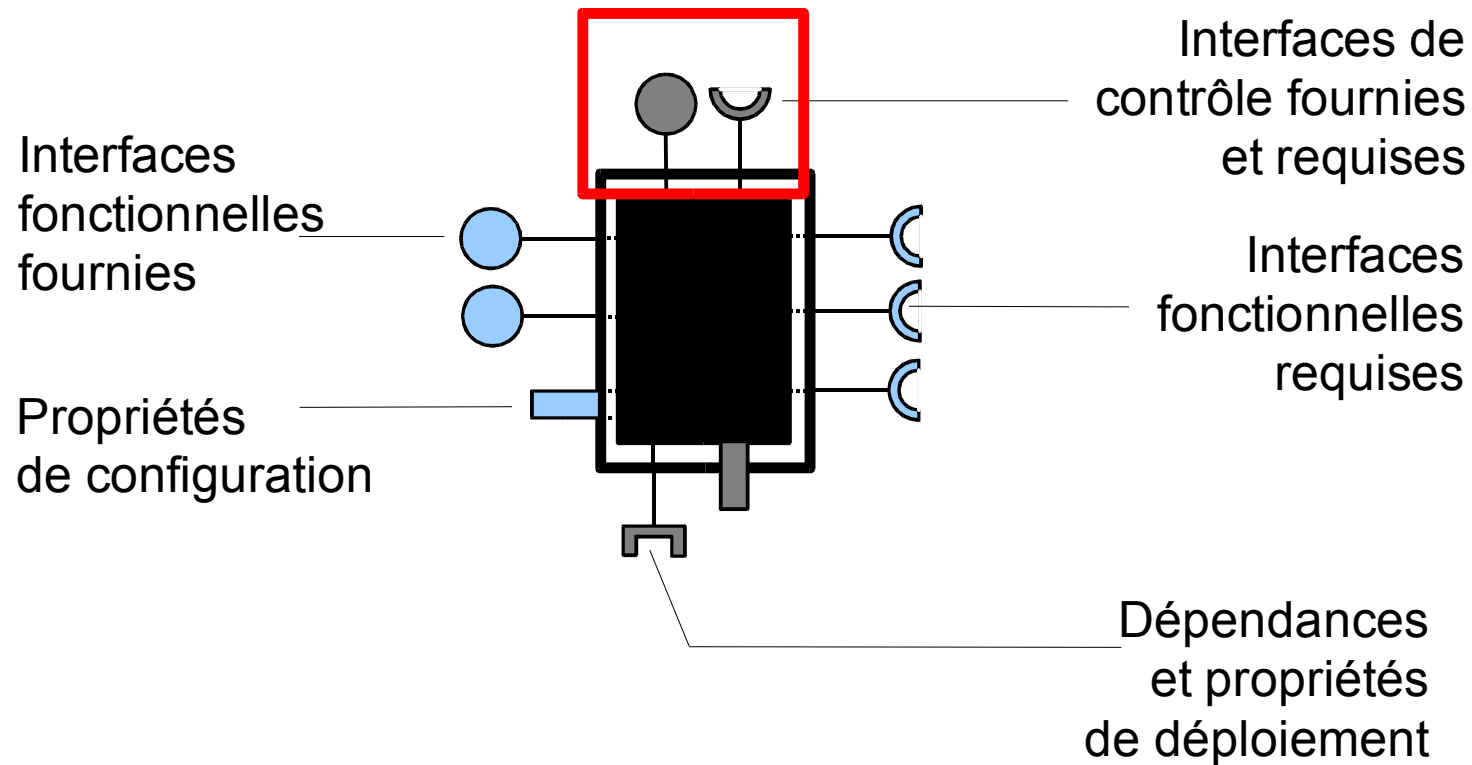
Éléments liés au déploiement



Remplies au moment du déploiement



Éléments liés à l'exécution

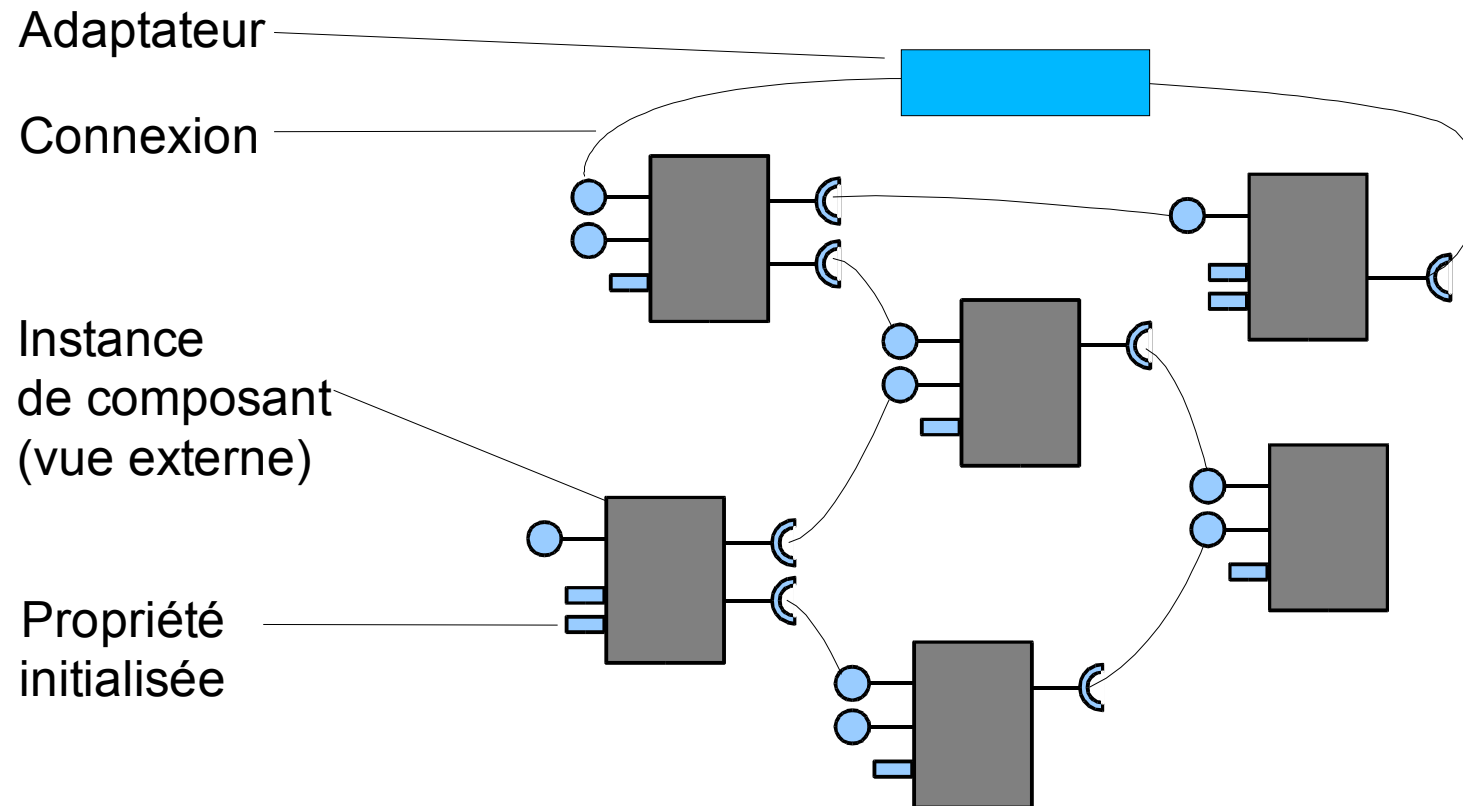


Gestion du cycle de vie des instances



Assemblage et composition

Assemblage réalisé à travers *la composition*.





Assemblage et composition

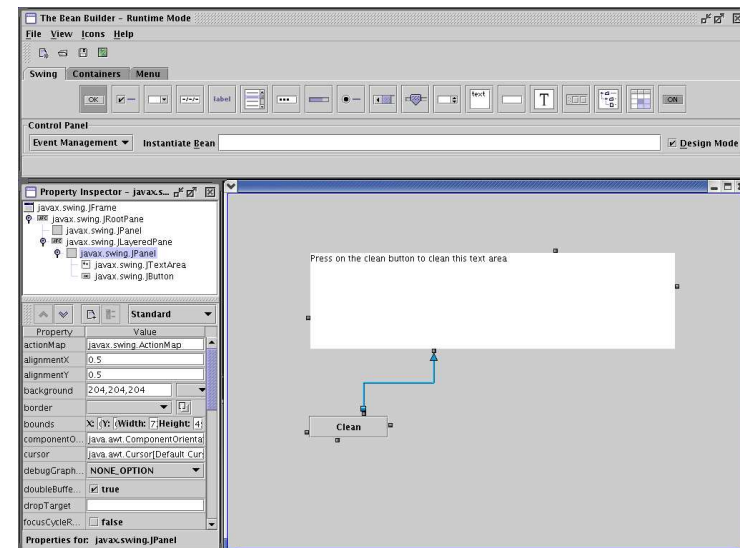
Types de *composition*.

```
System simple_cs = {  
  Component client = {Port sendRequest }  
  Component server = {Port receiveRequest}  
  Connector rpc = { Roles {caller, callee} }  
  Attachments : {  
    client.sendRequest to rpc.caller;  
    server.receiveRequest to rpc.callee }  
}
```

Déclaratif

```
Server server = (Server) Beans.instantiate  
(this.getClassLoader(),  
"org.examples.Server");  
Client client = (Client) Beans.instantiate  
(this.getClassLoader(),  
"org.examples.Client");  
server.add(client);
```

Impératif



Visuel



Assemblage et composition

Composition *hiérarchique*

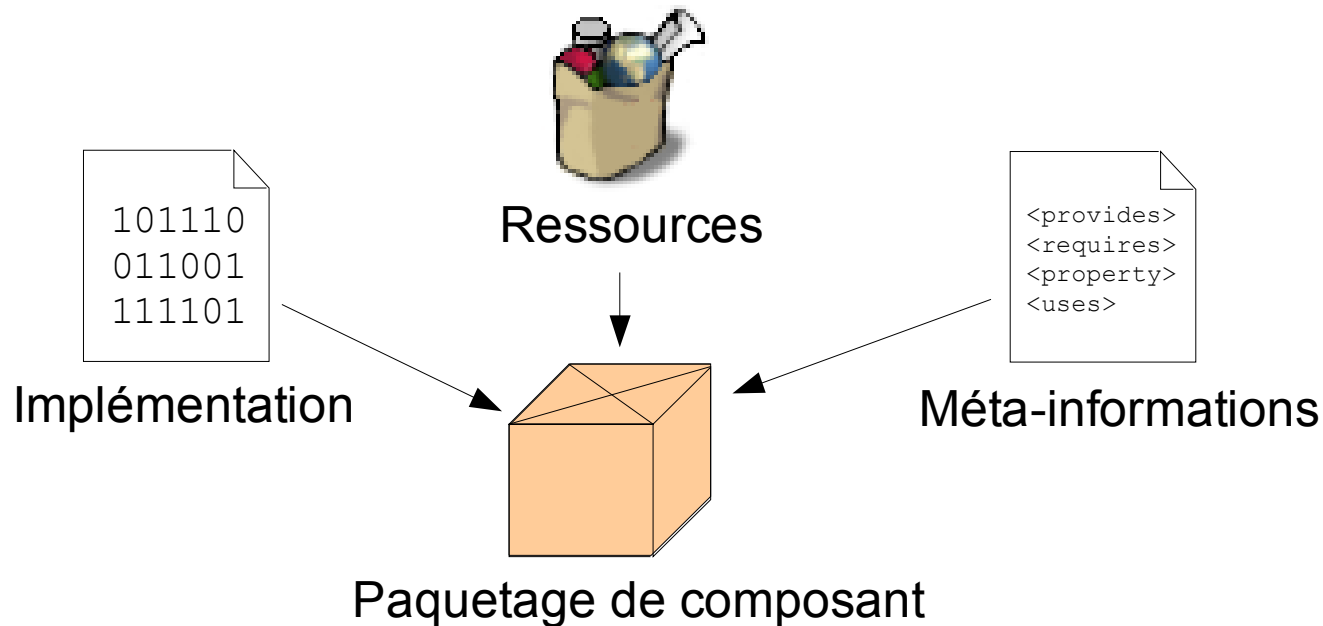




Paquetage de composant

Livraison et déploiement indépendants

- Paquetage contient tout ce qui est nécessaire au composant, sauf les dépendances explicites
- Binaire

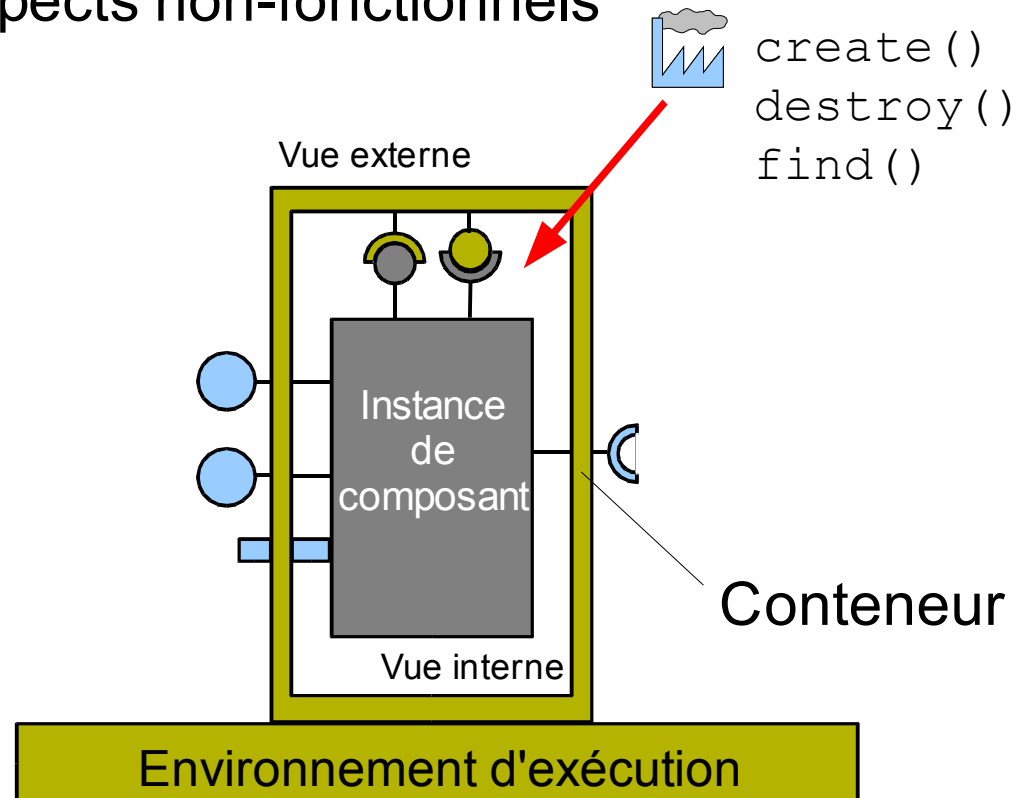




Fabriques et conteneurs

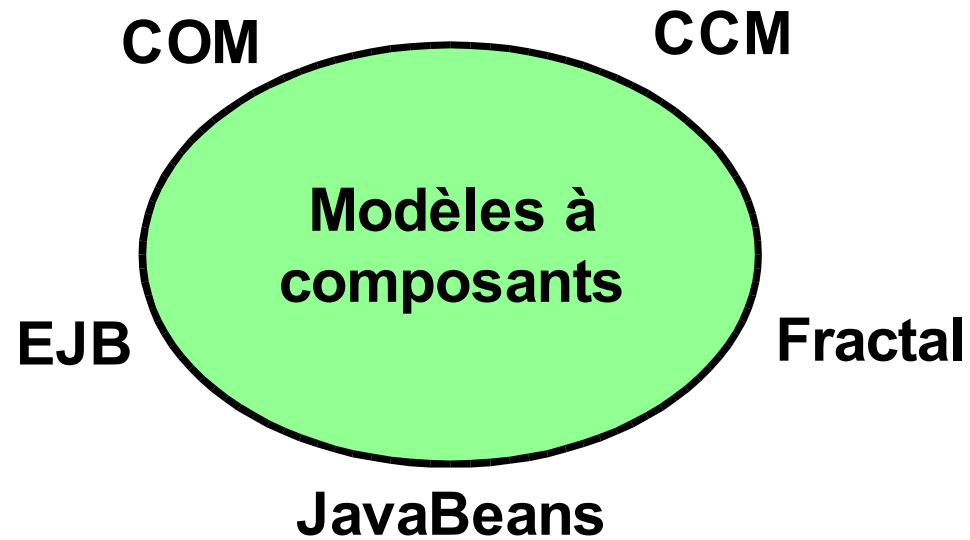
Instances créées à partir de fabriques et placées dans un conteneur

- Gestion du cycle de vie
- Gestion d'aspects non-fonctionnels





Modèles à composants



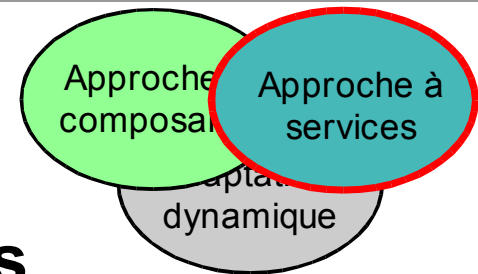


Plan de la thèse

Problématique

Fondations

- Approche à composants
- Approche à services
- Adaptation dynamique



Modèle à composants orienté services

Réalisation

Évaluations

Conclusions et perspectives



Approche à services

Services : fonctionnalités réutilisables

- Décrits de façon contractuelle
- Fournisseurs de service substituables

Assemblage réalisé à partir de descripteurs

- Informations syntaxiques + sémantiques

Intégration *tardive* des fournisseurs

- Mécanismes de découverte dynamique
- Patron d'interaction de l'approche à services

Disponibilité dynamique est une hypothèse



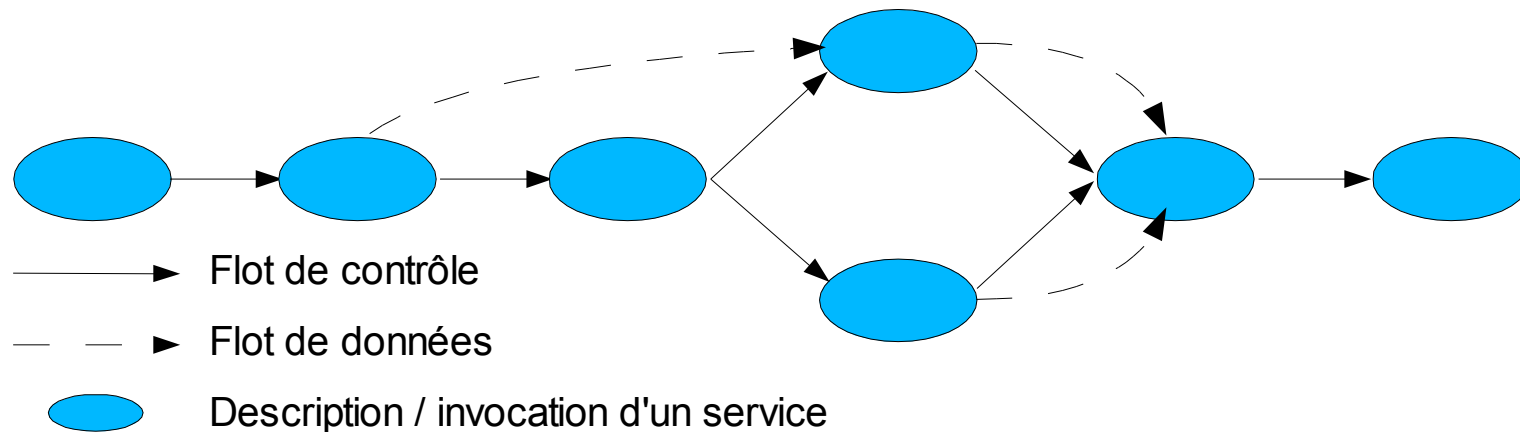
Composition de services

Composition basée sur descripteurs de service

- Fournisseurs intégrés avant ou durant exécution

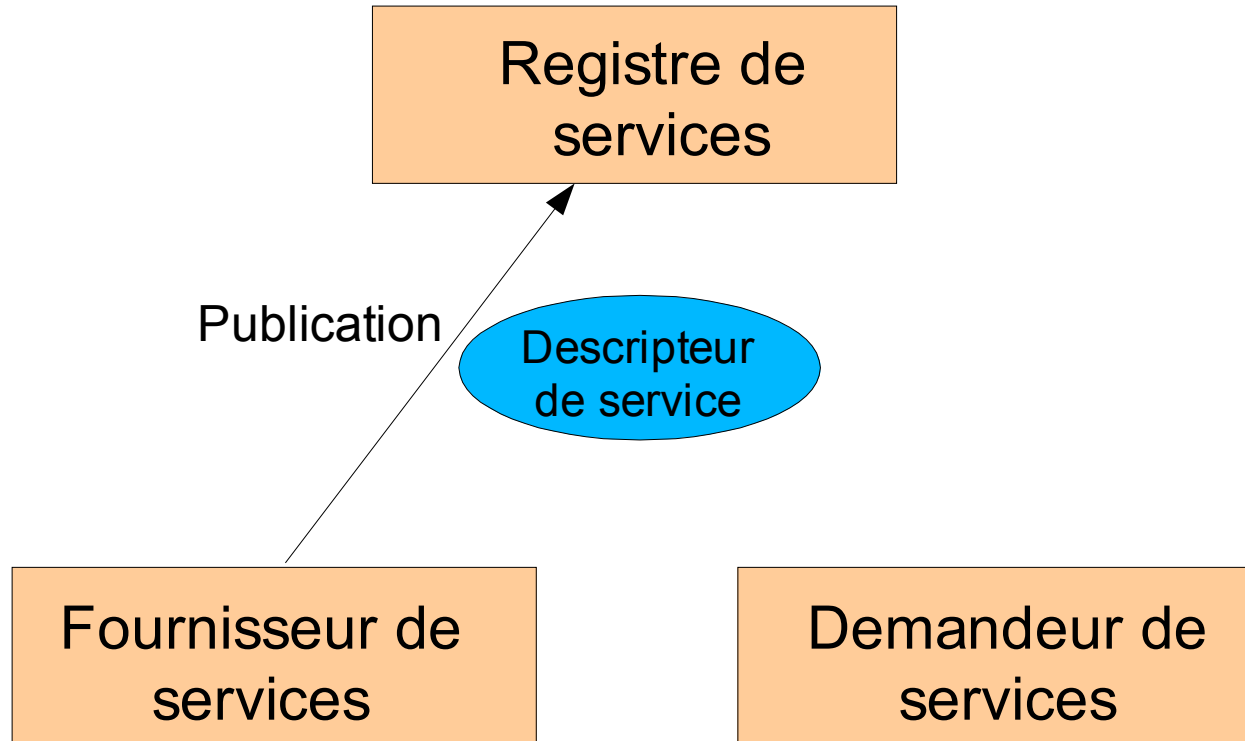
Approche à flot favorisée

- Invocation séquentielle
- Transfert de données



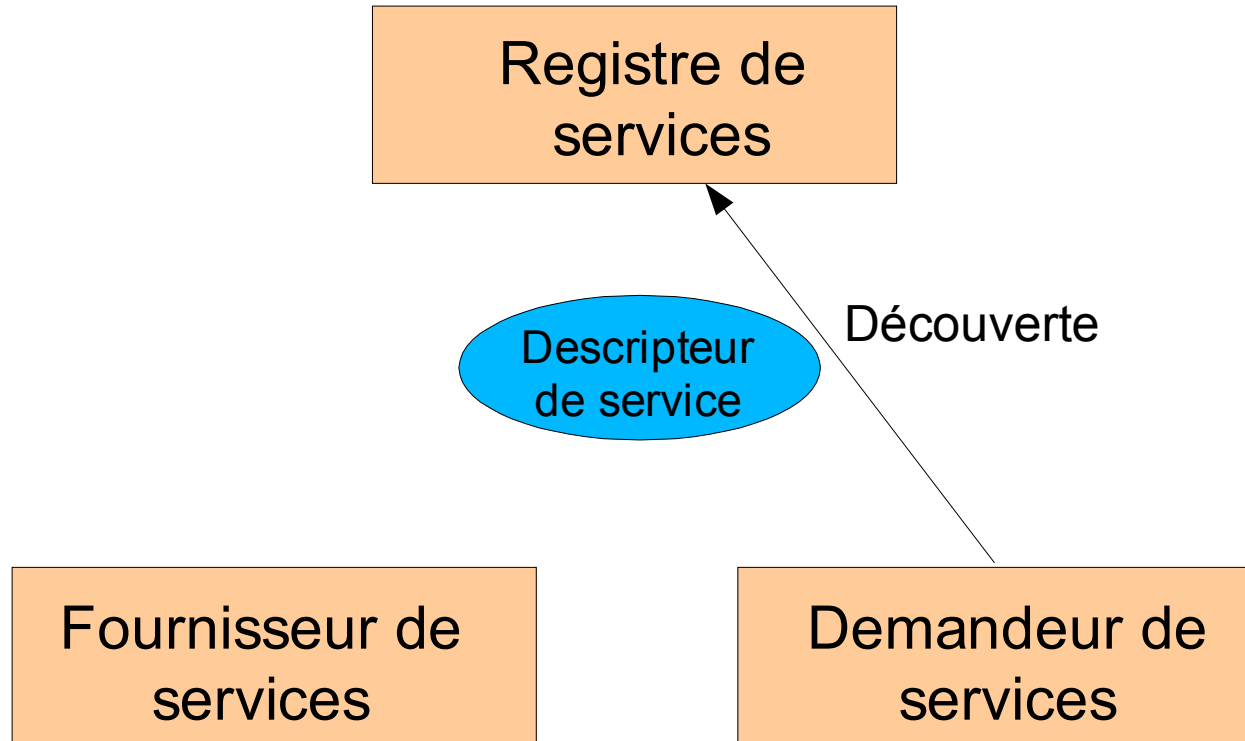


Patron d'interaction



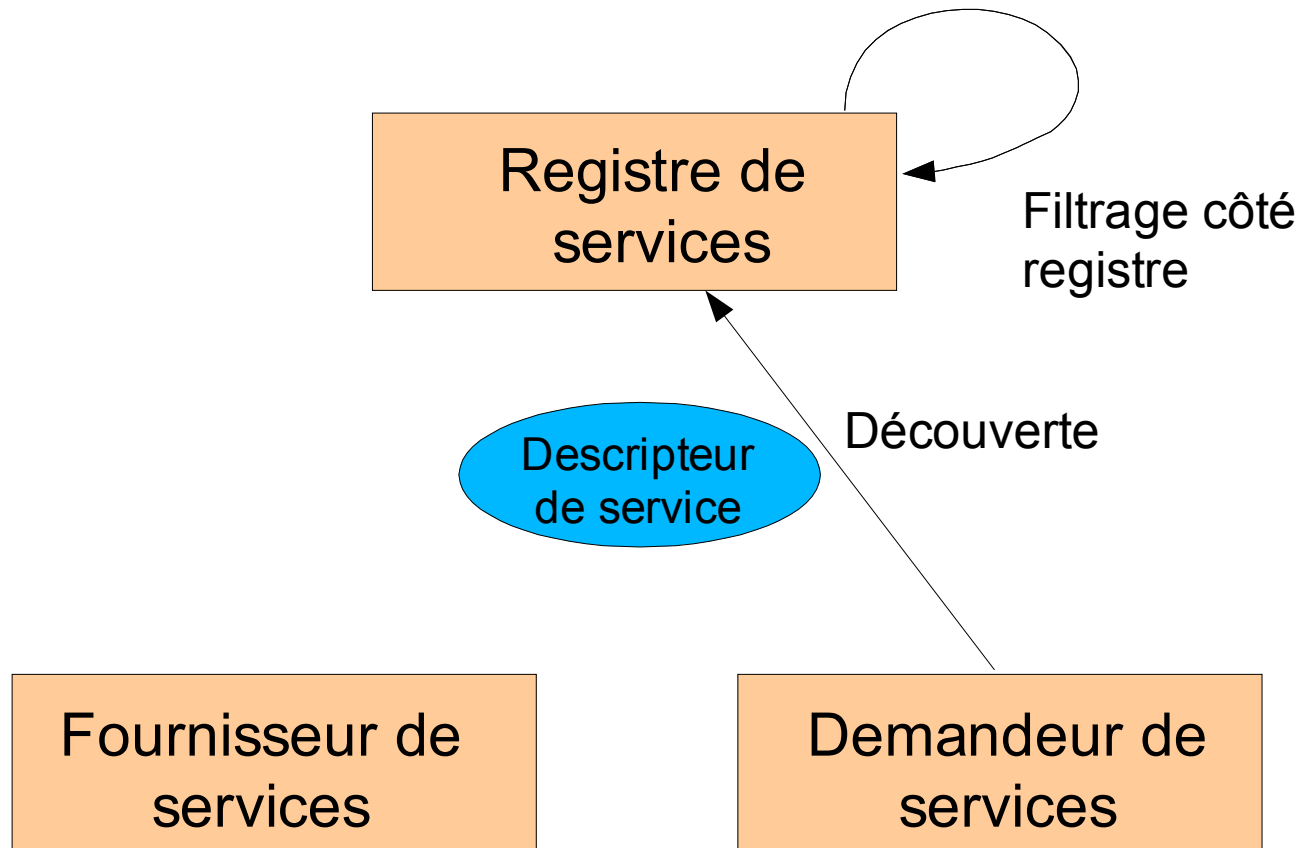


Patron d'interaction



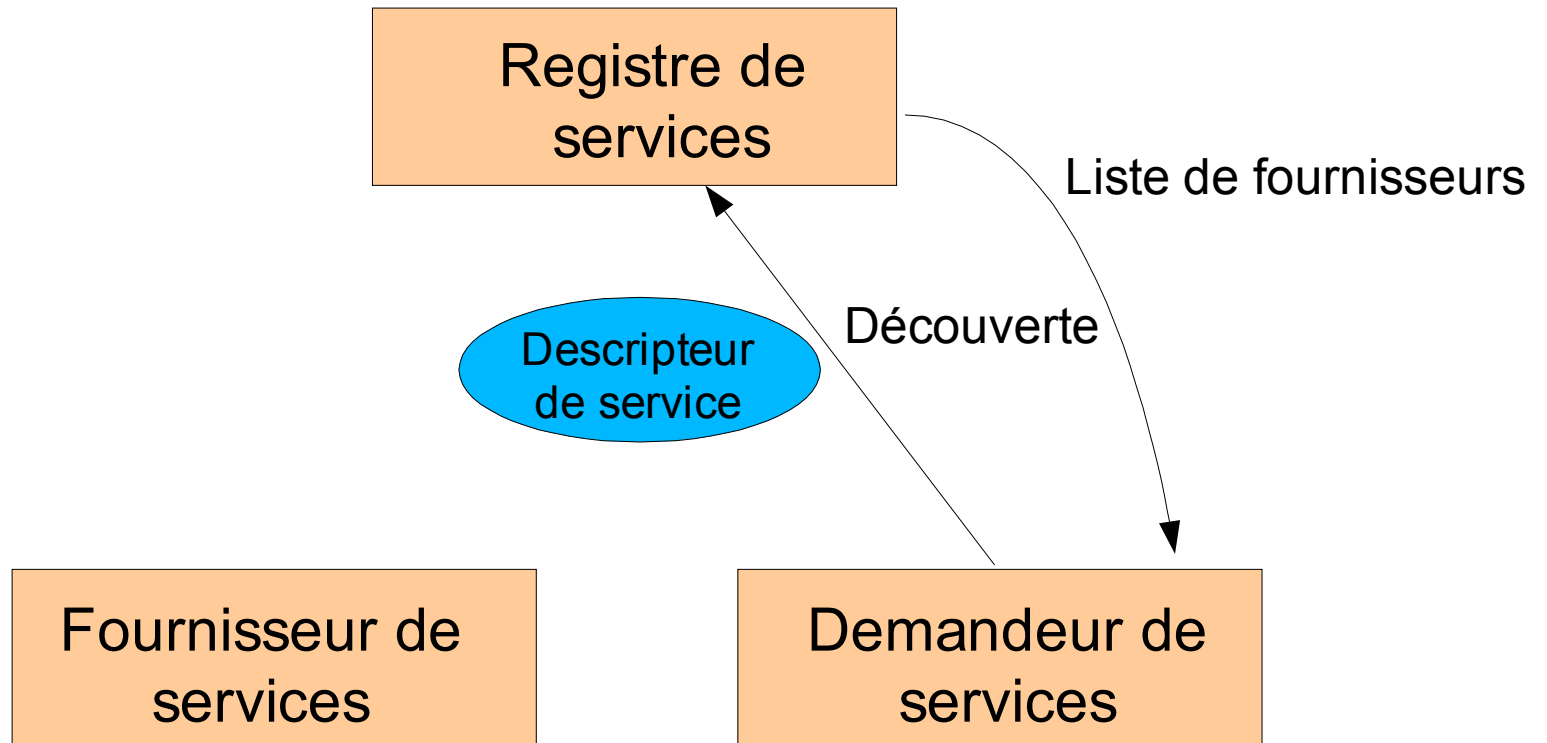


Patron d'interaction



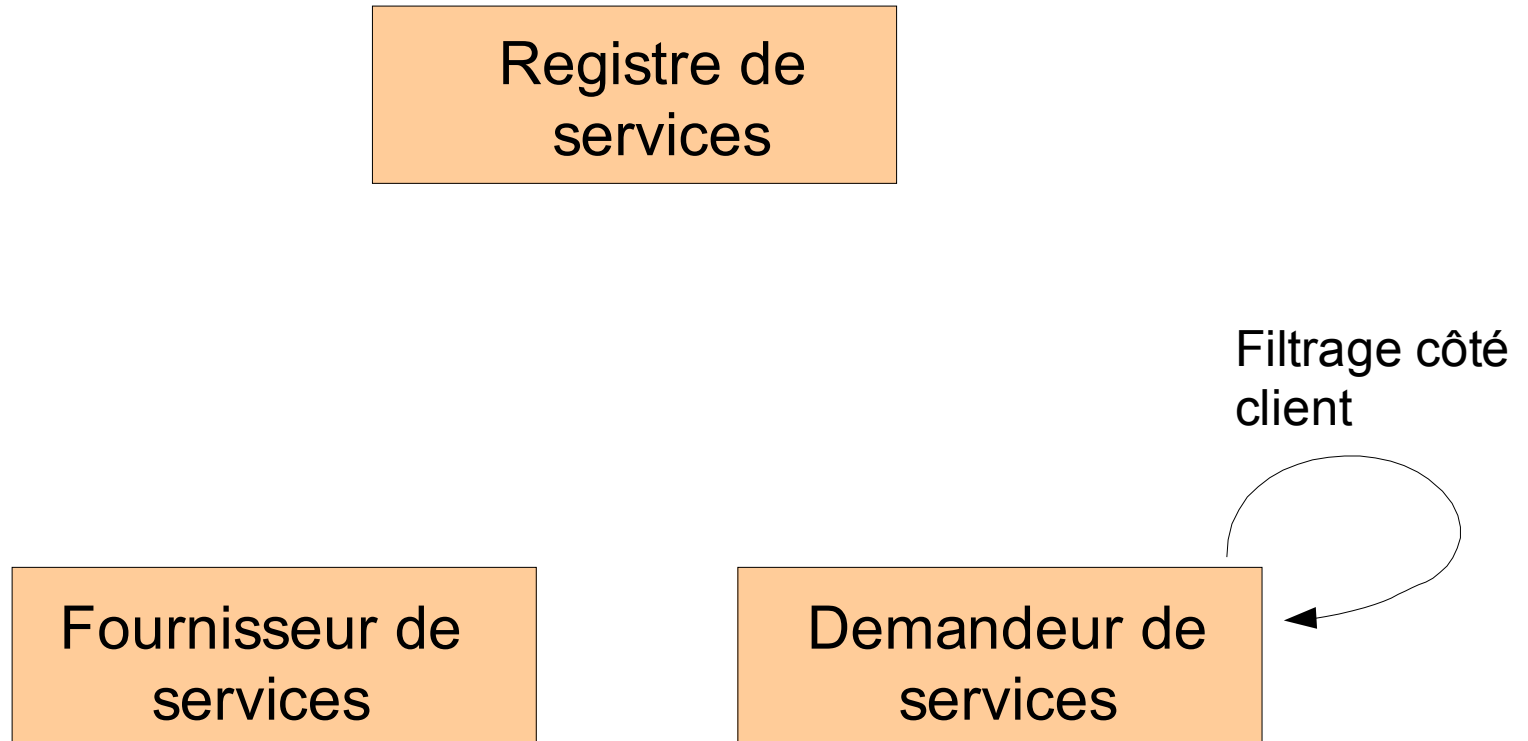


Patron d'interaction



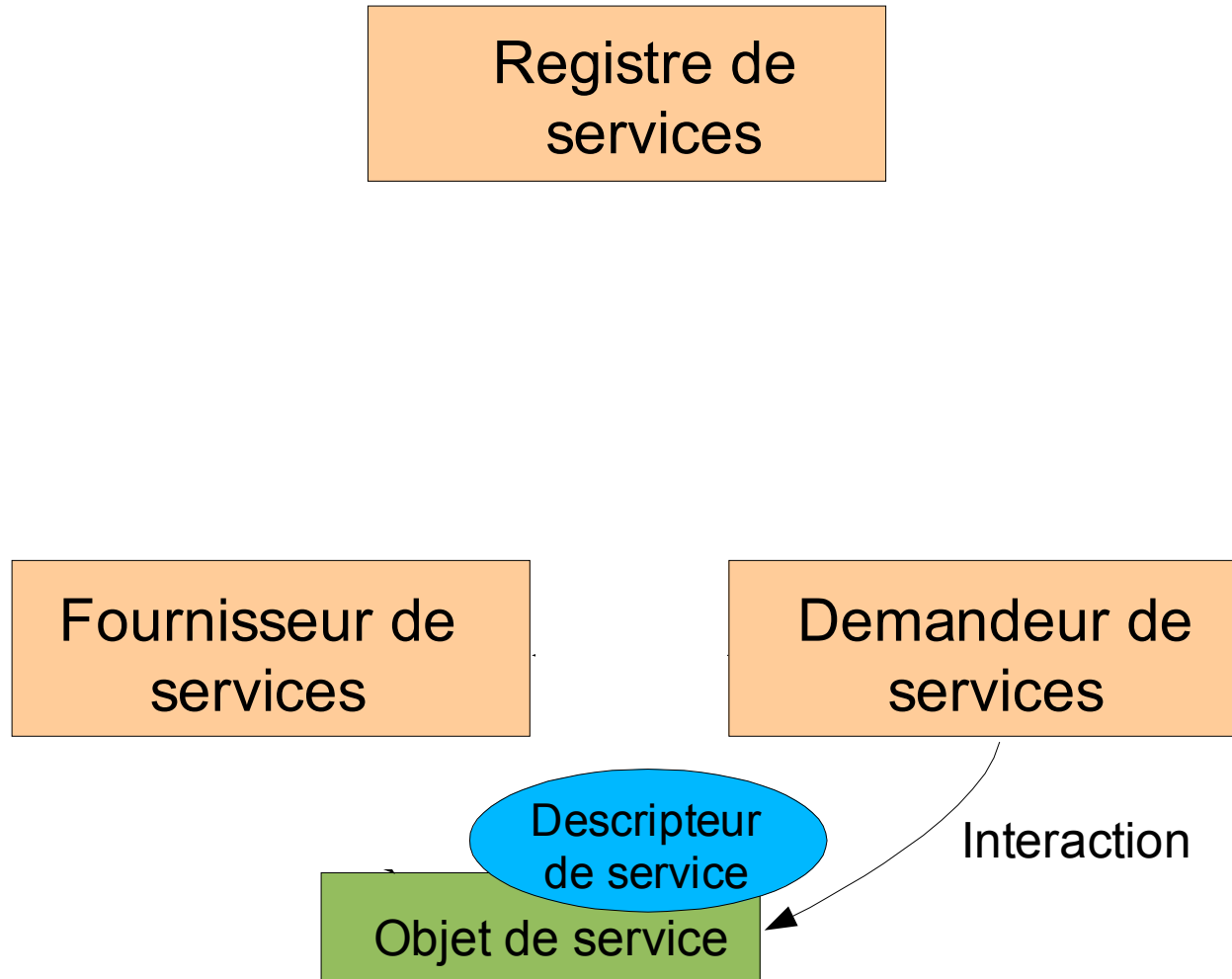


Patron d'interaction



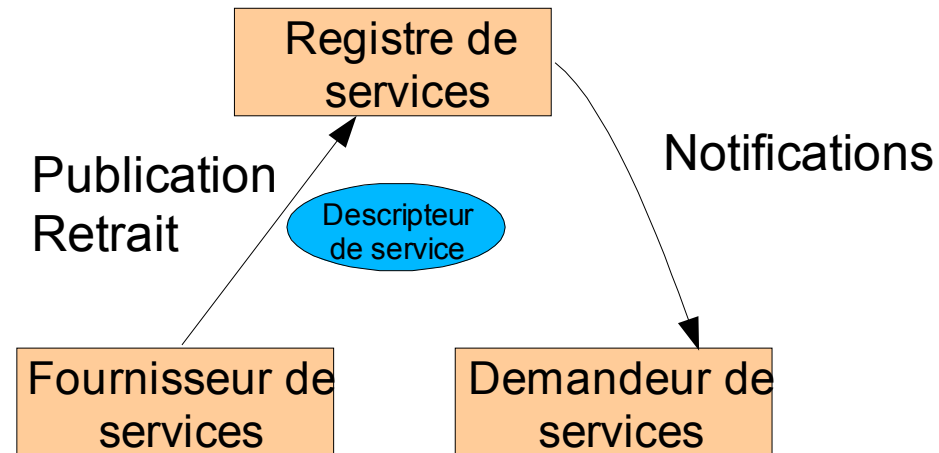


Patron d'interaction





Disponibilité dynamique



Mécanismes de notification

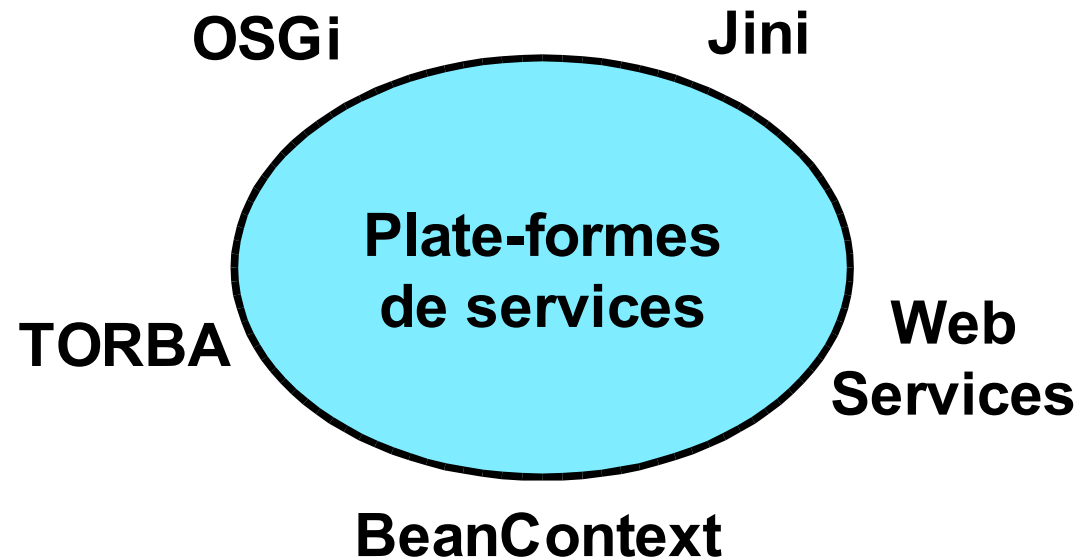
- Arrivée, départ, changement des services

Comment gérer les changements?

- A la charge des demandeurs
- Mécanismes de transactions dans composition à flots



Plate-formes de services





Plan de la thèse

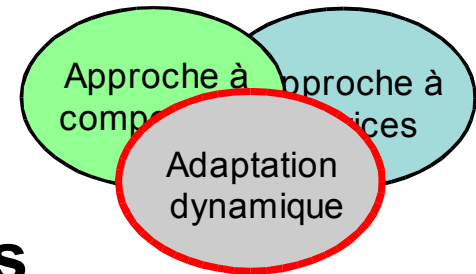
Problématique

Fondations

- Approche à composants
- Approche à services
- Adaptation dynamique



Modèle à composants orienté services



Réalisation

Évaluations

Conclusions et perspectives



Adaptation dynamique

Adaptation d'une application pendant l'exécution

- Application adaptative
- Auto-adaptation

Logique d'adaptation

- Supervision
- Adaptation



Adaptation à travers *reconfiguration dynamique*

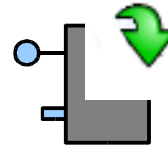
- Modification de l'architecture d'une application à l'exécution



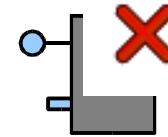
Reconfiguration dynamique

Opérations primitives

- create
- destroy
- bind
- unbind
- set
- move



`create()`



`destroy()`



`bind()`



`unbind()`

A travers une représentation architecturale

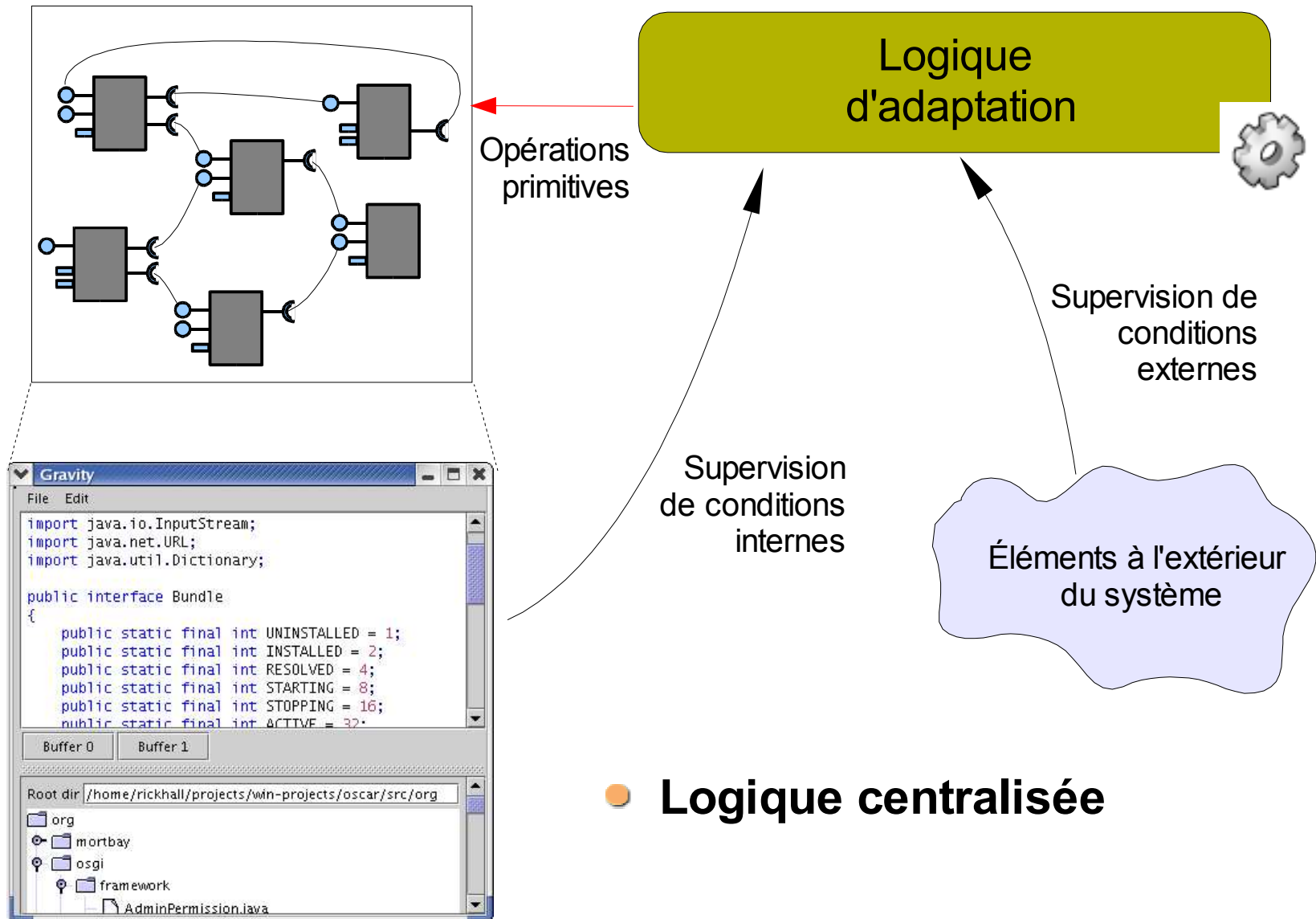
- Participation des composants

Préservation de la consistance

- État correct après changements



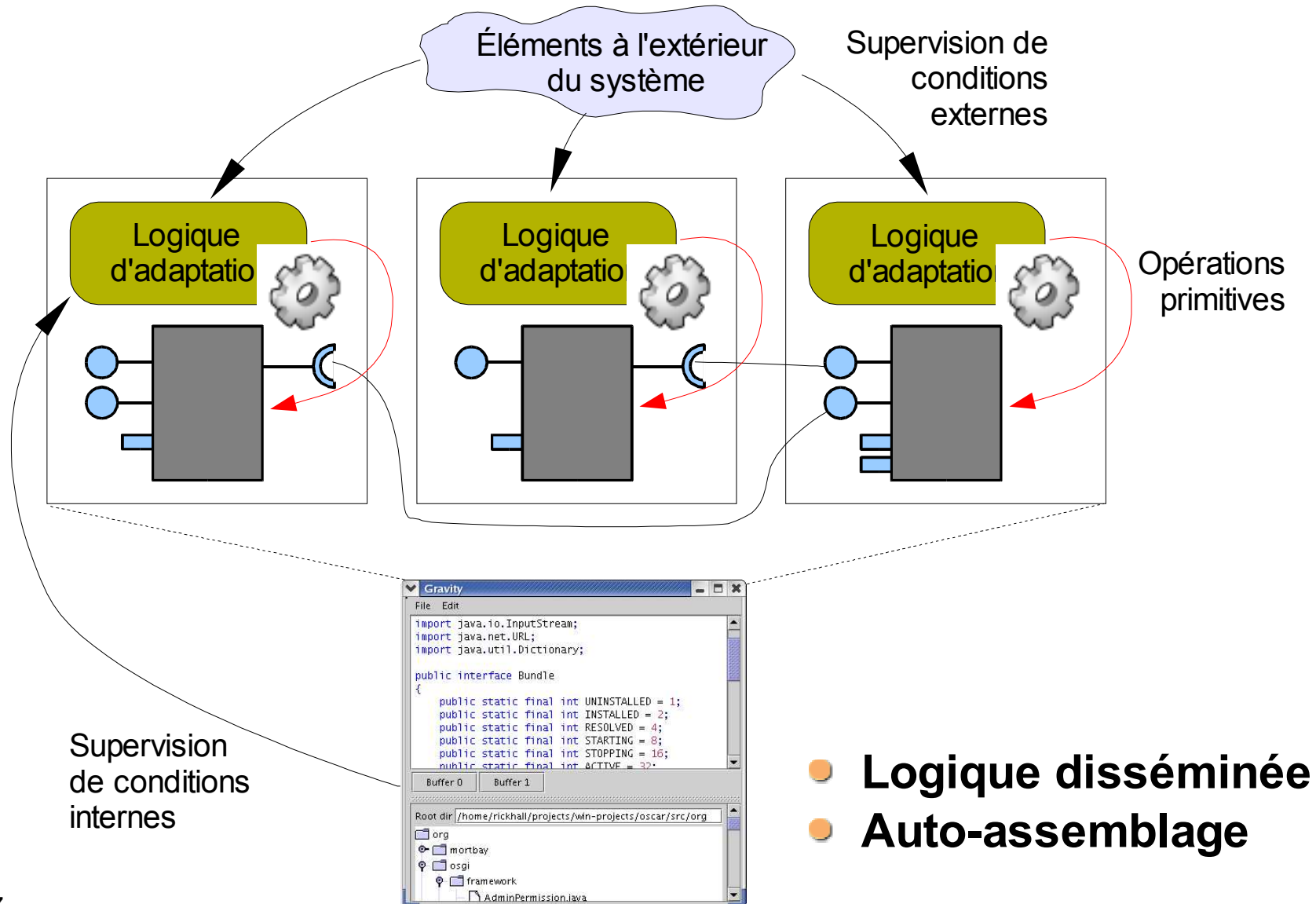
Adaptation dynamique



- **Logique centralisée**



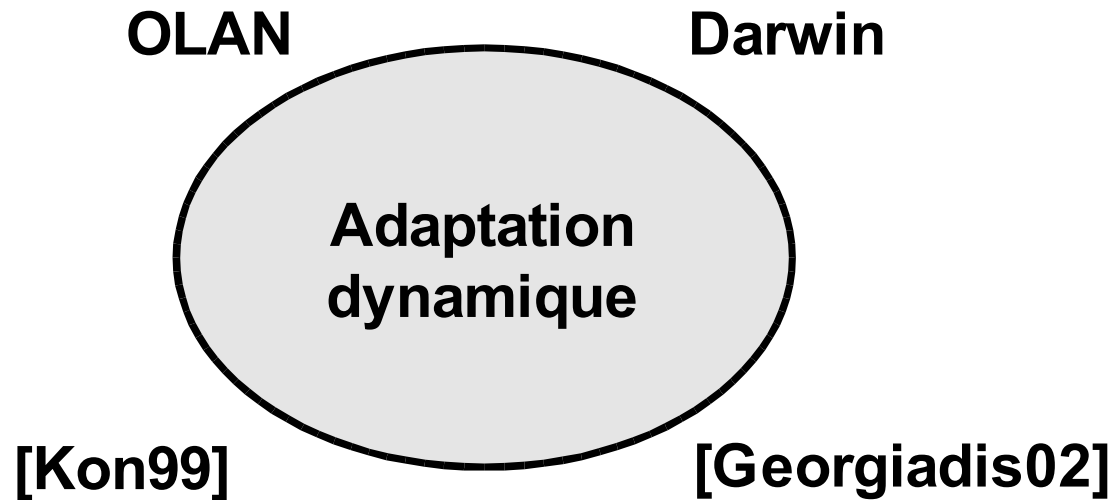
Adaptation dynamique



- Logique disséminée
- Auto-assemblage



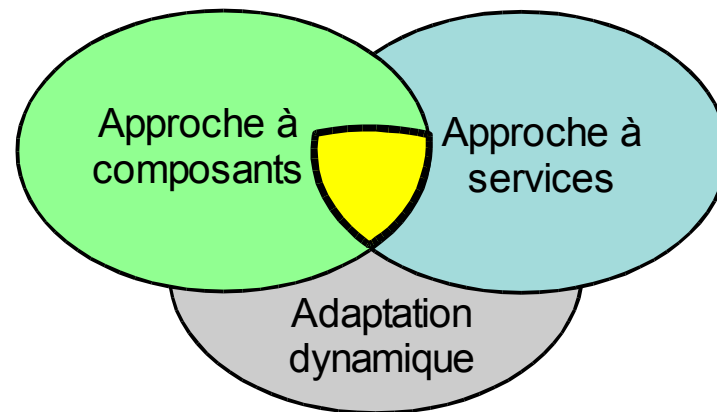
Adaptation dynamique





Modèle à composants orienté services

- **Fondation**



- **Découverte dynamique**
- **Intégration tardive**
- **Disponibilité dynamique**

- **Auto-adaptation**
- **Logique disséminée**
- **Adaptation gérée par environnement d'exécution**



Plan de la thèse

Problématique

Fondations

Modèle à composants orienté services

- Composants à services
- Gestion des instances
- Assemblage et évolution d'une application
- Types d'instances
- Imprévisibilité et espaces de résolution
- Gestion de composition
- Descripteur de composition

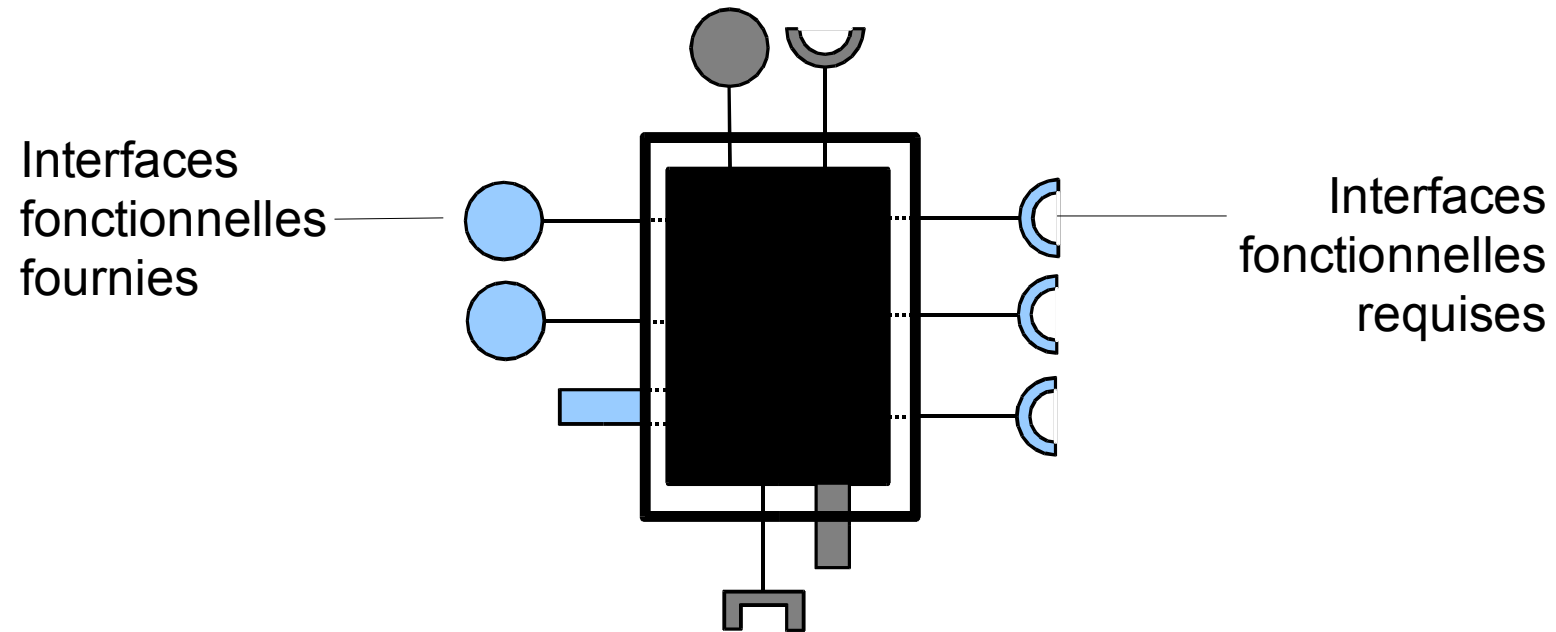
Réalisation

Évaluations

Conclusions et perspectives

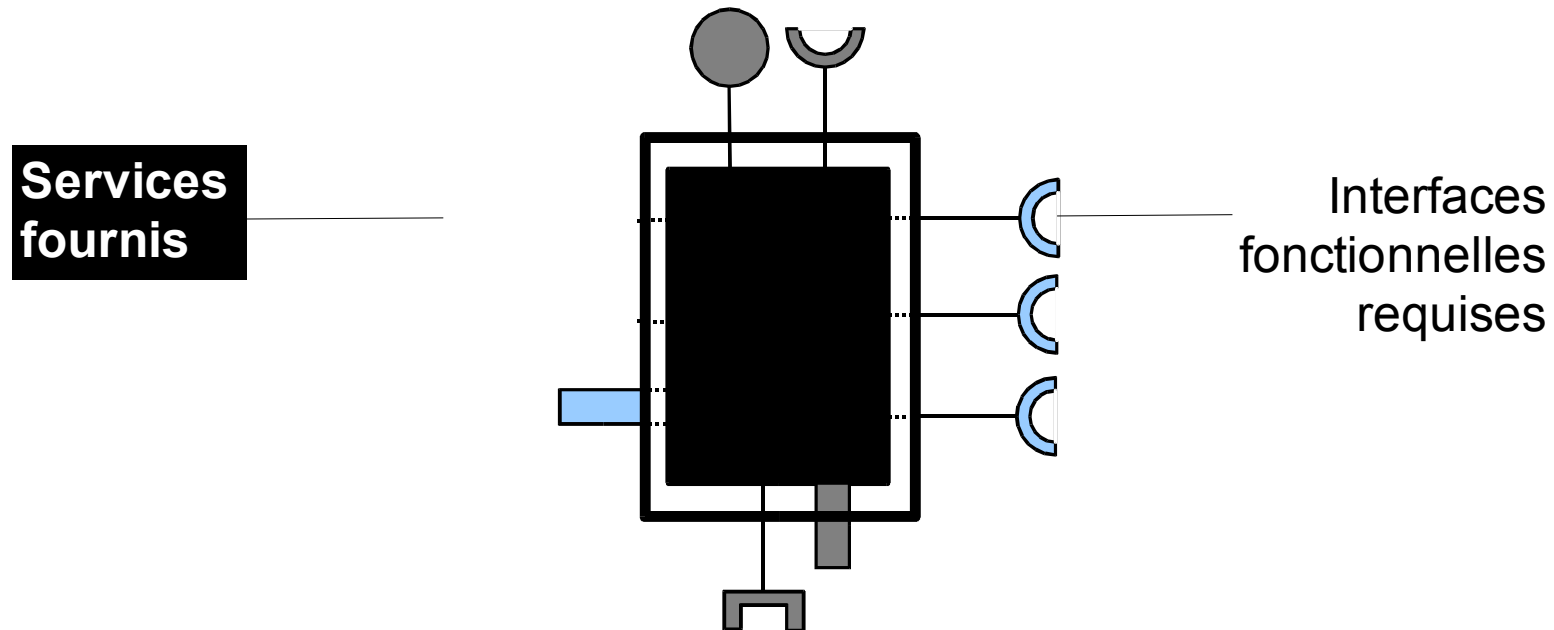


Composant à services





Composant à services

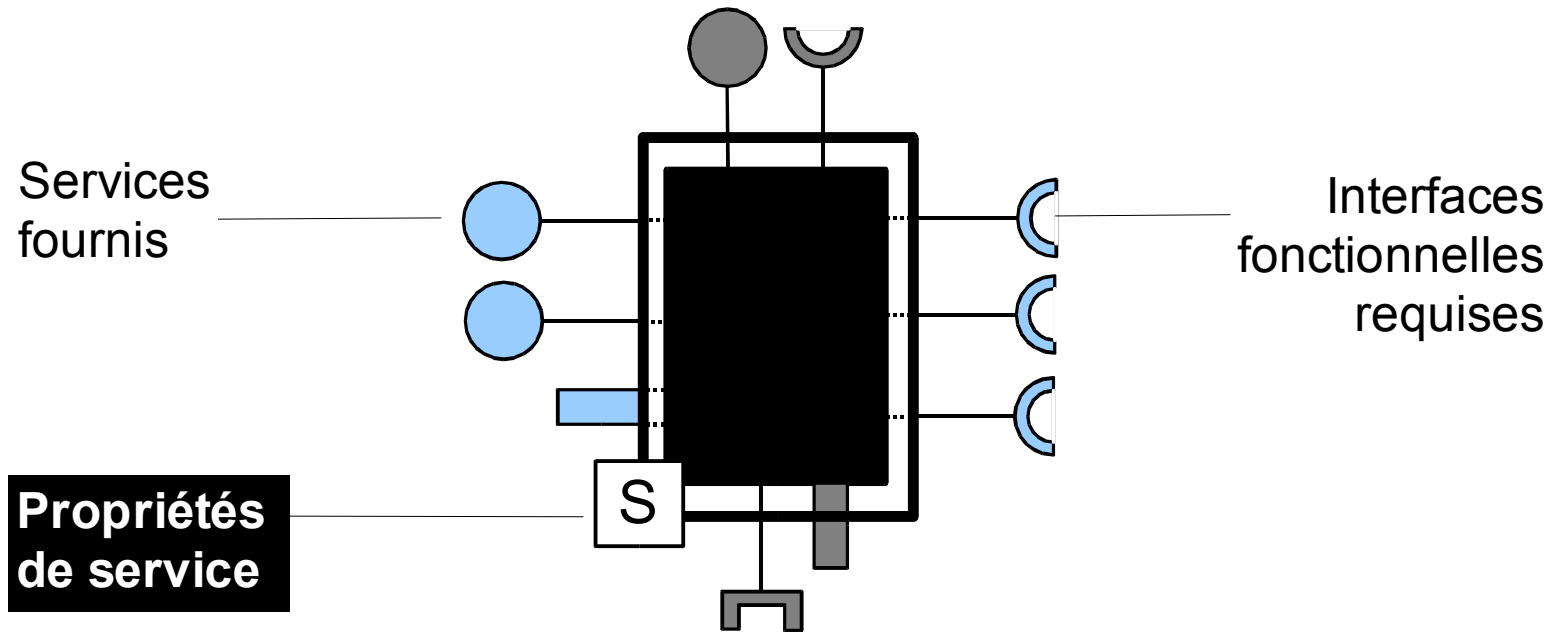


Publication dans un registre de services

- Services fournis par les instances du composant
- Disponibles dynamiquement par rapport à *validité*



Composant à services

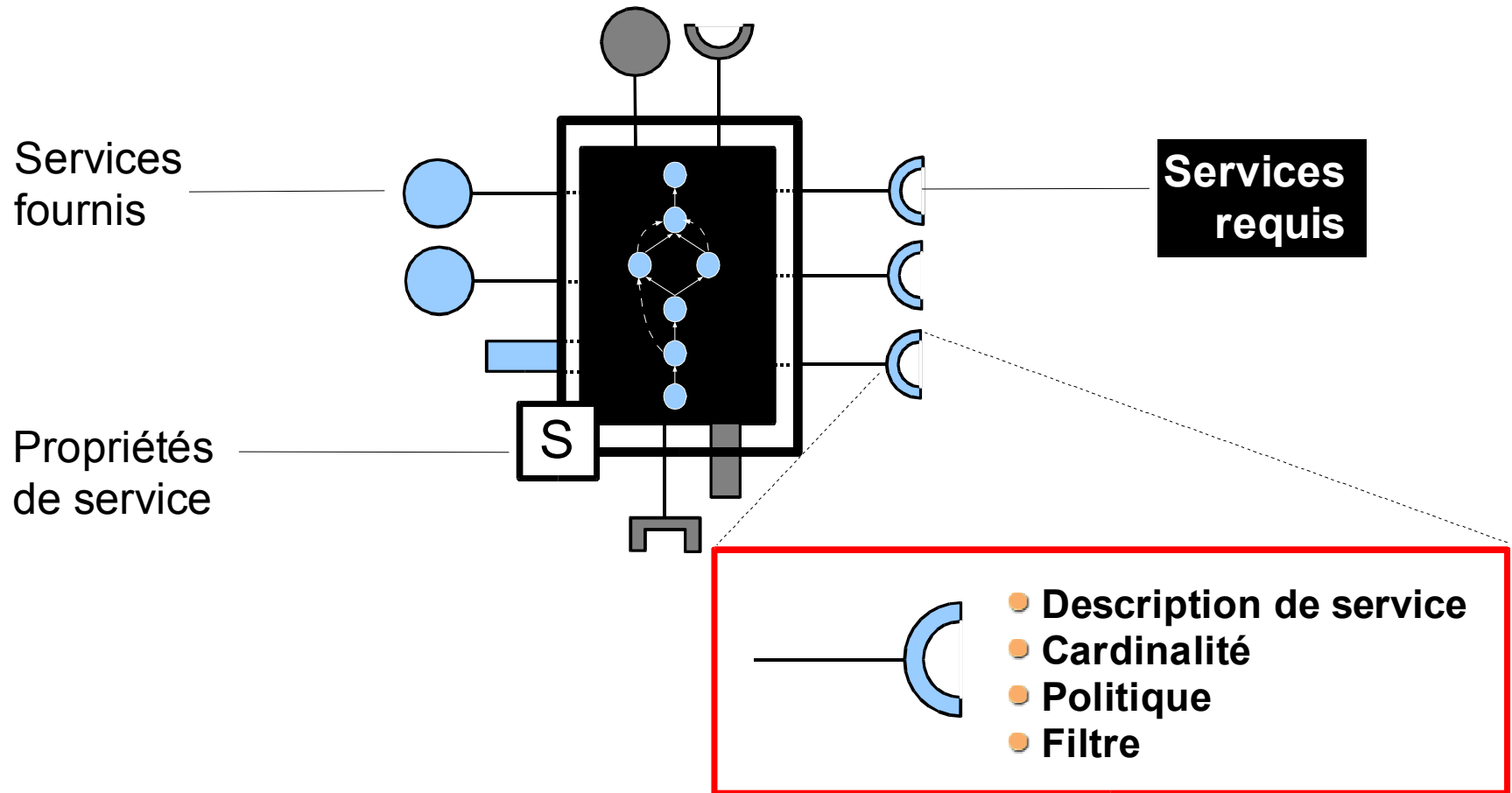


Distinguent fournisseurs d'un même service

- <clé, valeur>



Composant à services






Plan de la thèse

Problématique

Fondations

Modèle à composants orienté services

- Composants à services
-  ● Gestion des instances
- Assemblage et évolution d'une application
- Types d'instances
- Imprévisibilité et espaces de résolution
- Gestion de composition
- Descripteur de composition

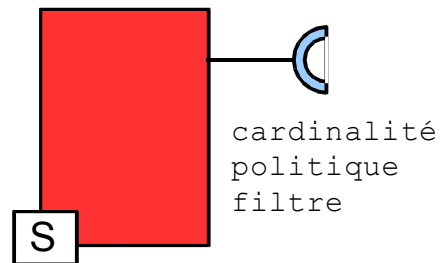
Réalisation

Évaluations

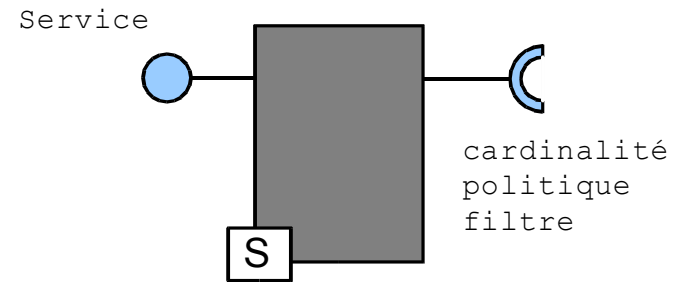
Conclusions et perspectives



Validité des instances



Instance *invalide*



Instance *valide*

Création intentionnelle

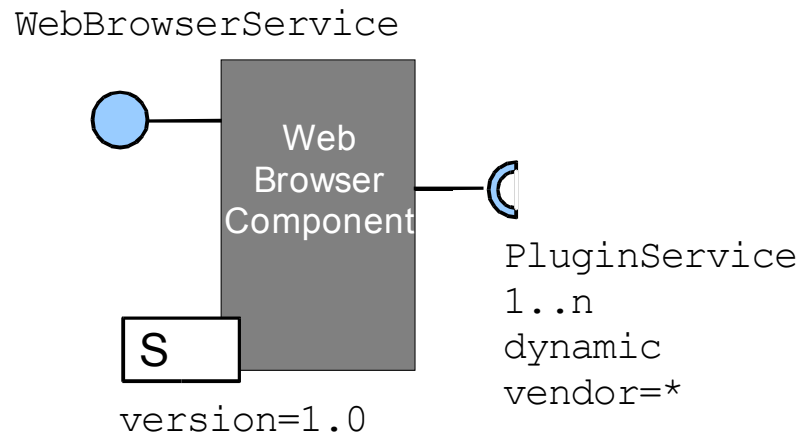
- Création ne garantit pas la validité
- Validité change pendant l'exécution

Validité

- par rapport aux services requis



Services requis



Cardinalité

- Optionnelle, obligatoire
- Simple, multiple

Politique

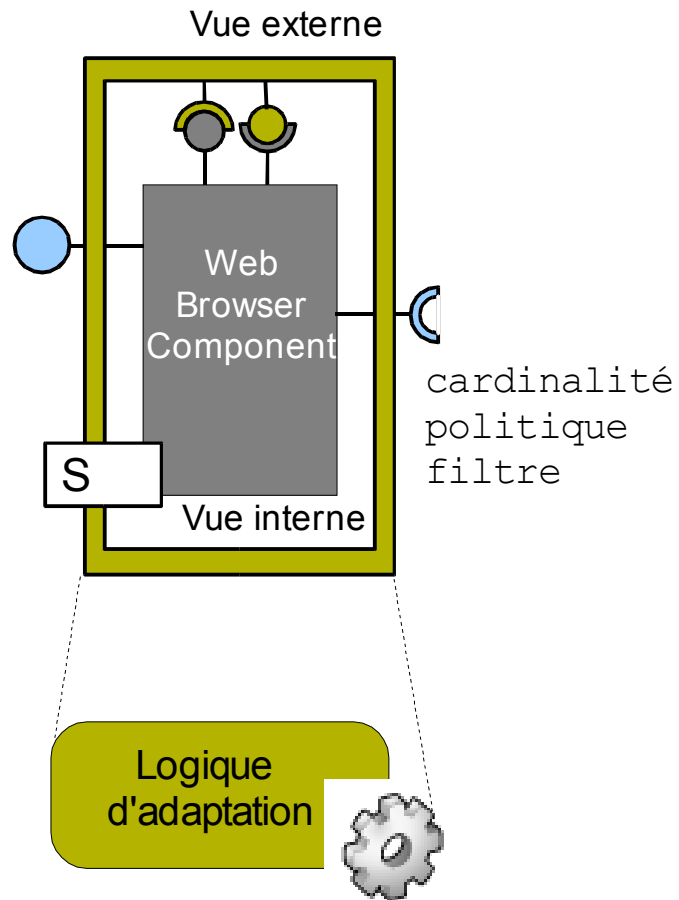
- Statique, dynamique

Filtre

- Par rapport aux propriétés de service



Gestionnaire d'instance



Conteneur indépendant

- Créé avec instance

Résponsabilités

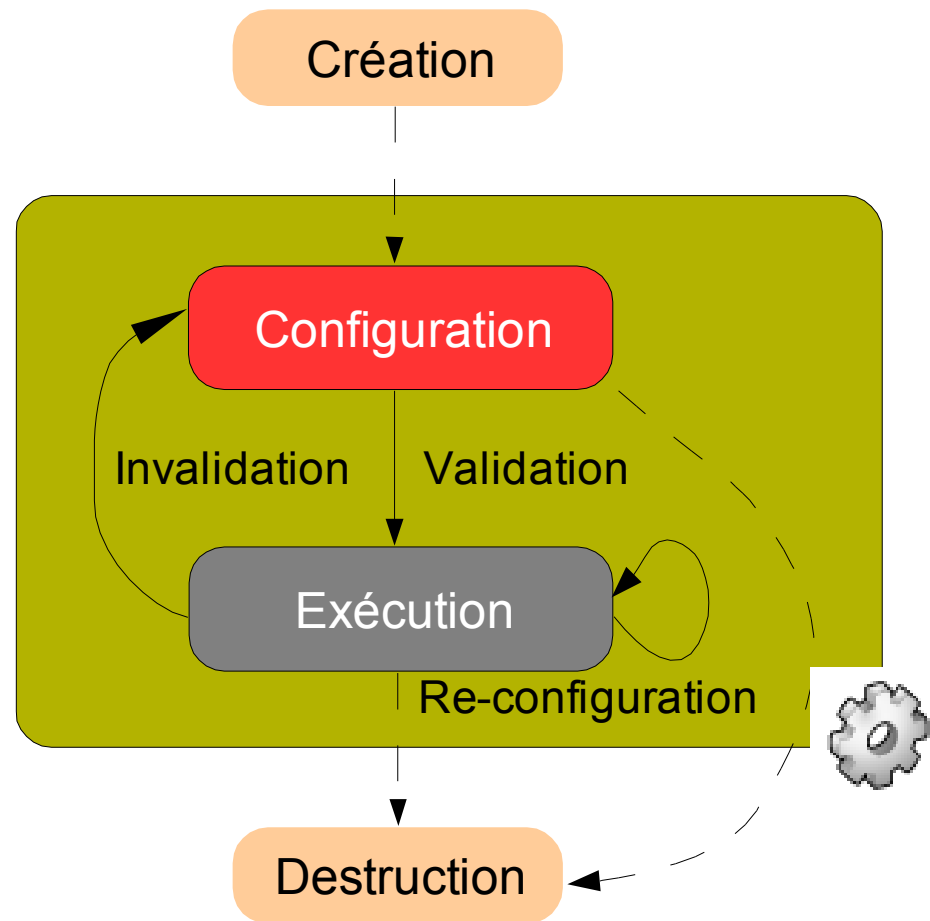
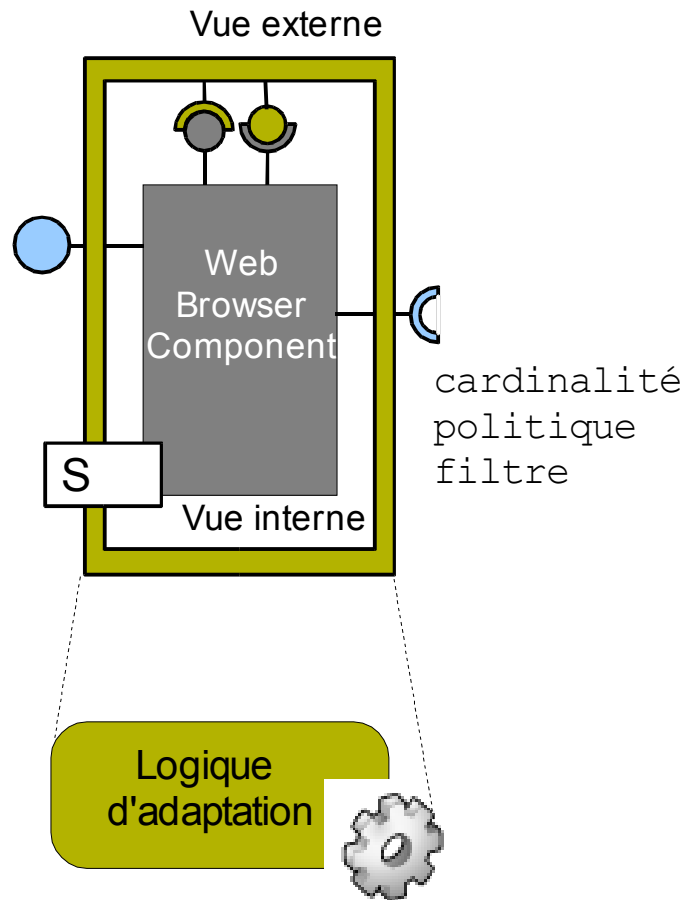
- Publication de services
- Gestion des dépendances
- Maintien de la validité

Gestion des dépendances

- Supervision des changements
 - Reconfigurations
- Opérations `bind` et `unbind`

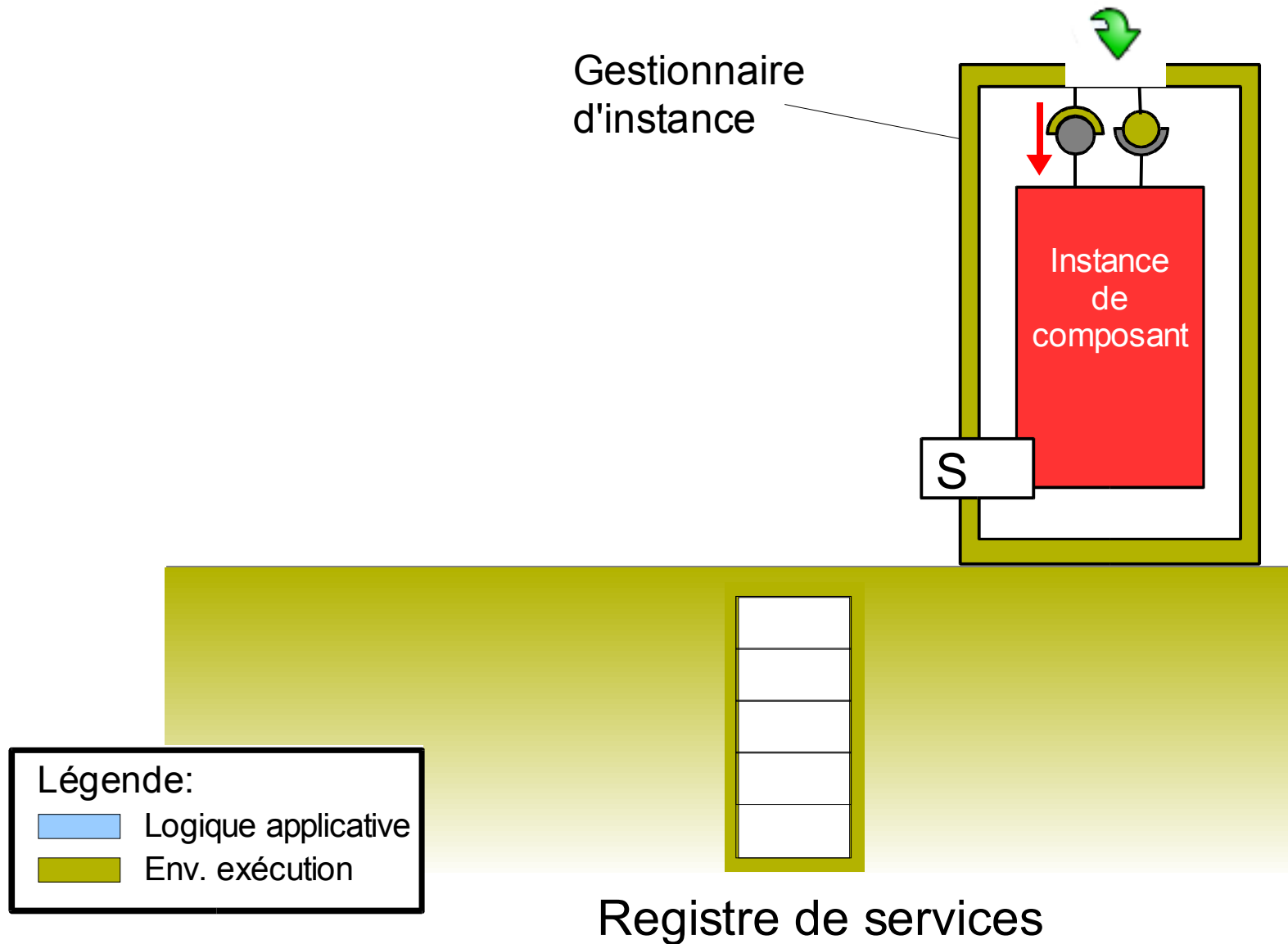


Gestionnaire d'instance



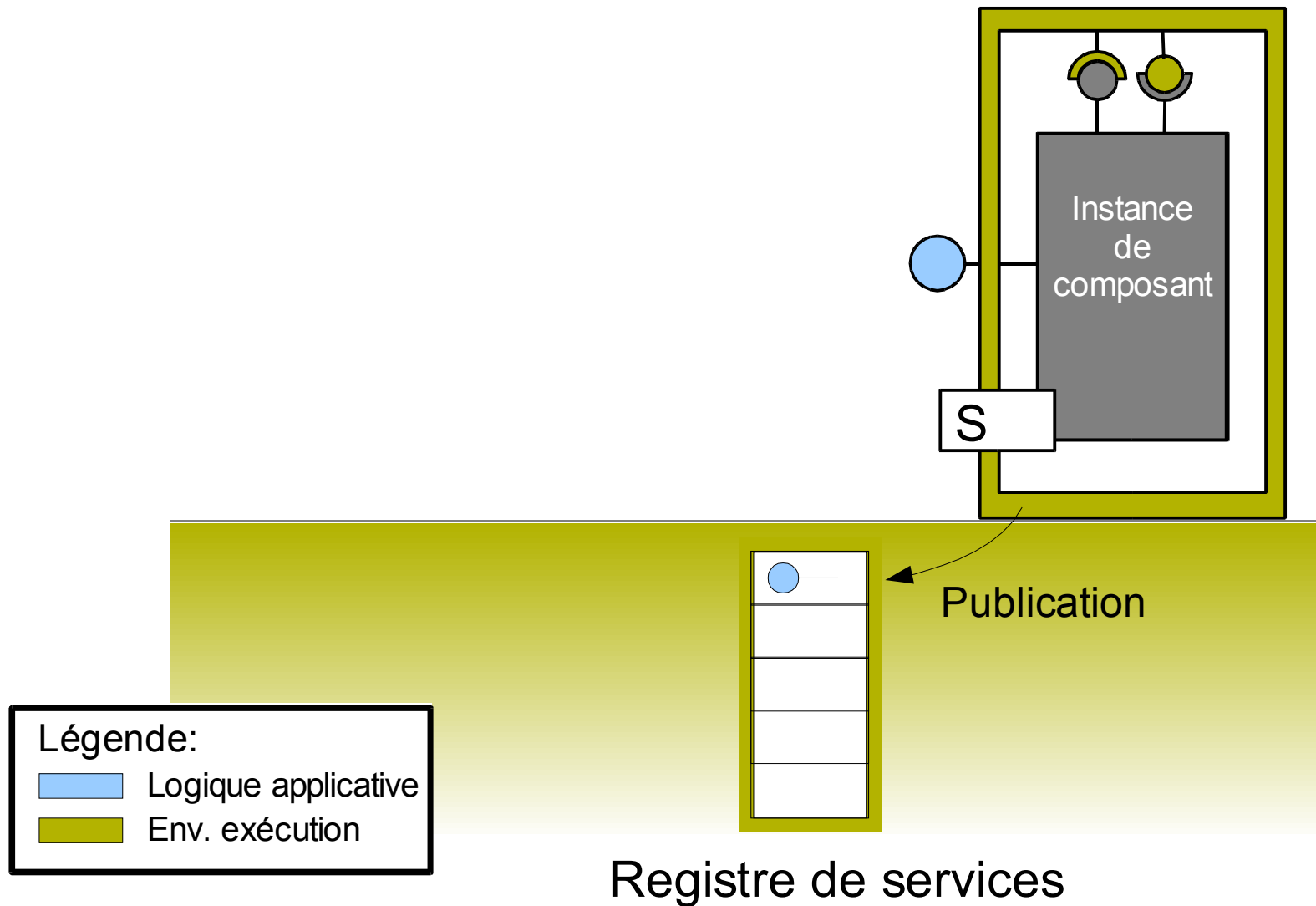


Gestion des instances



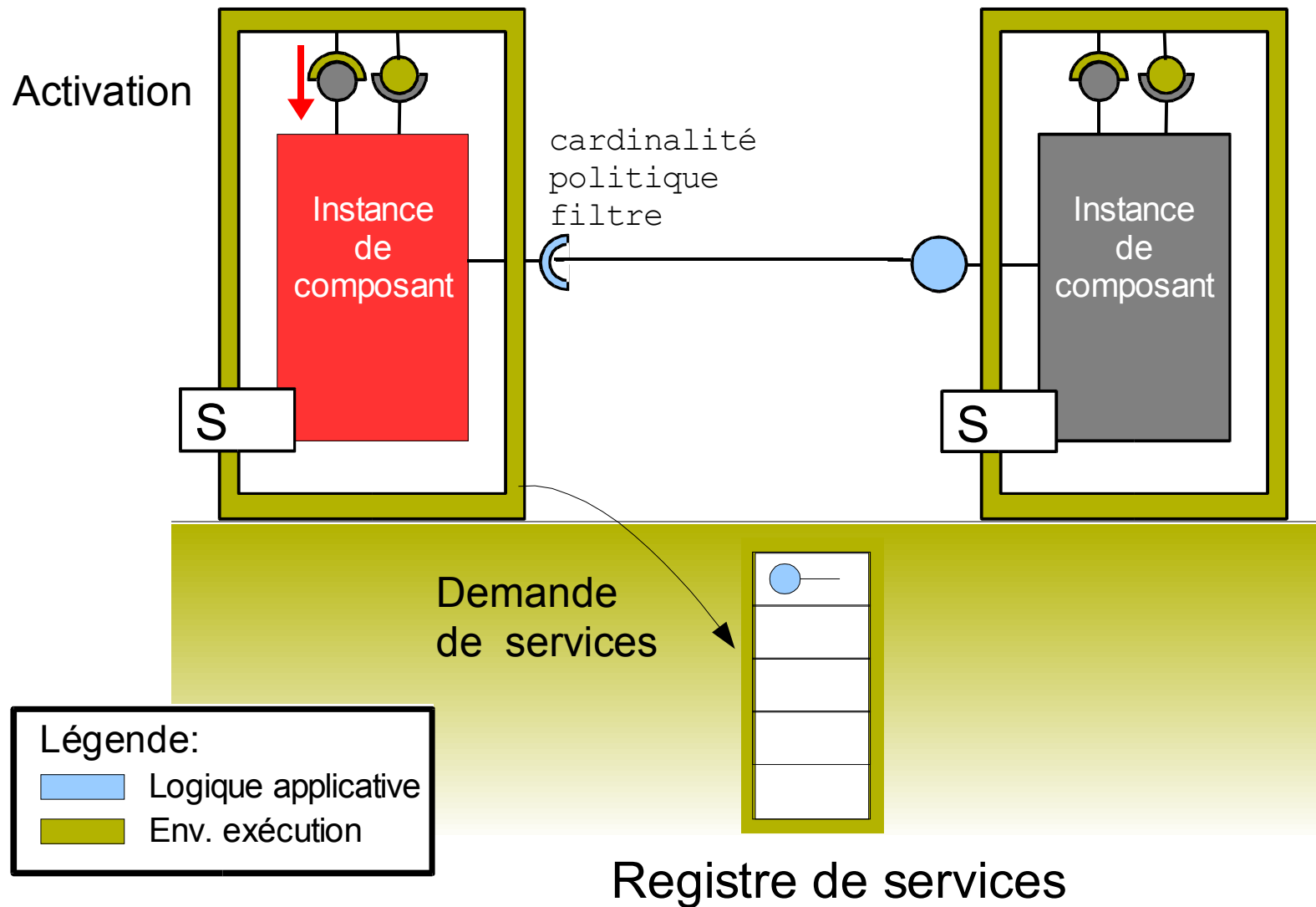


Gestion des instances



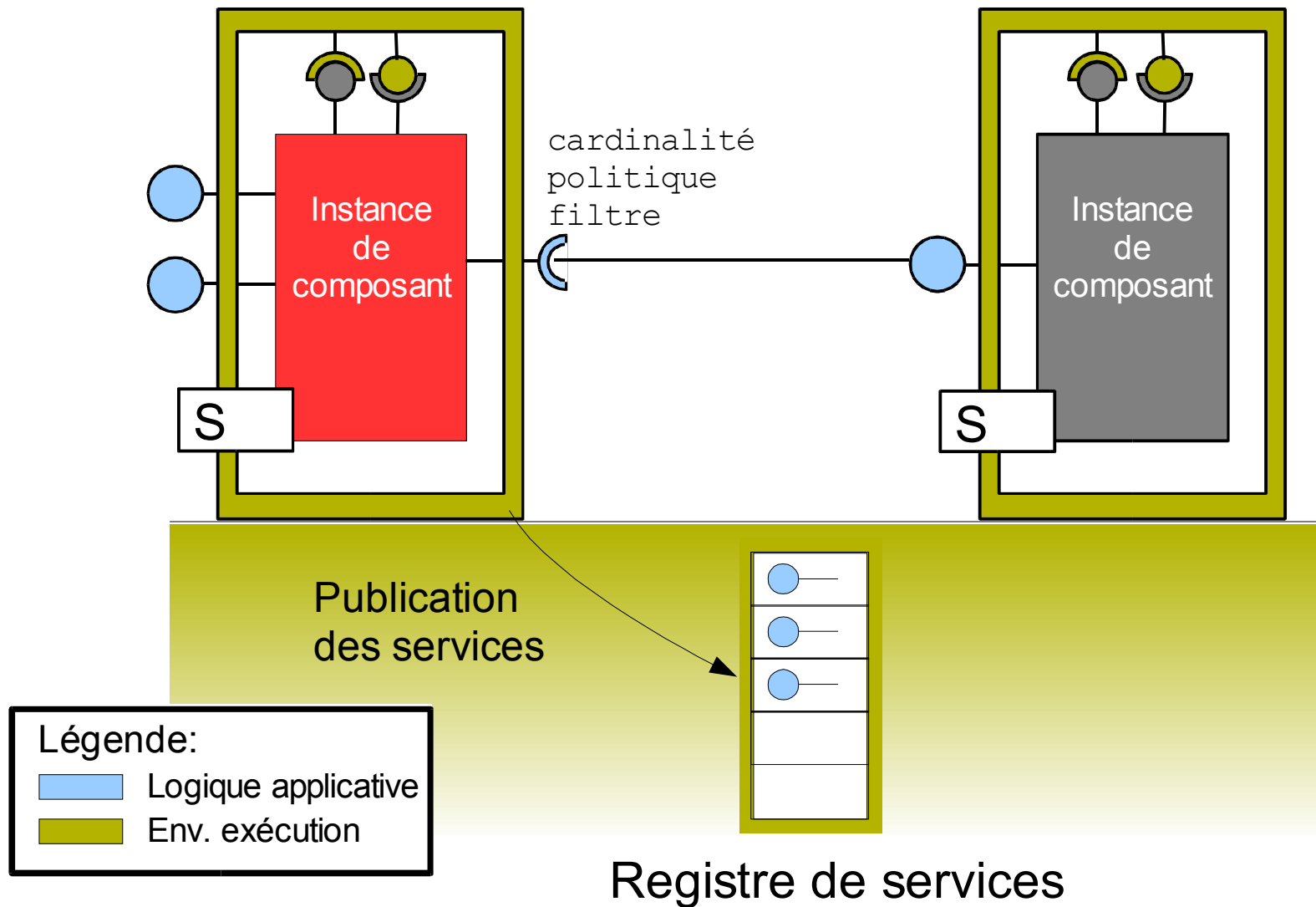


Gestion des instances



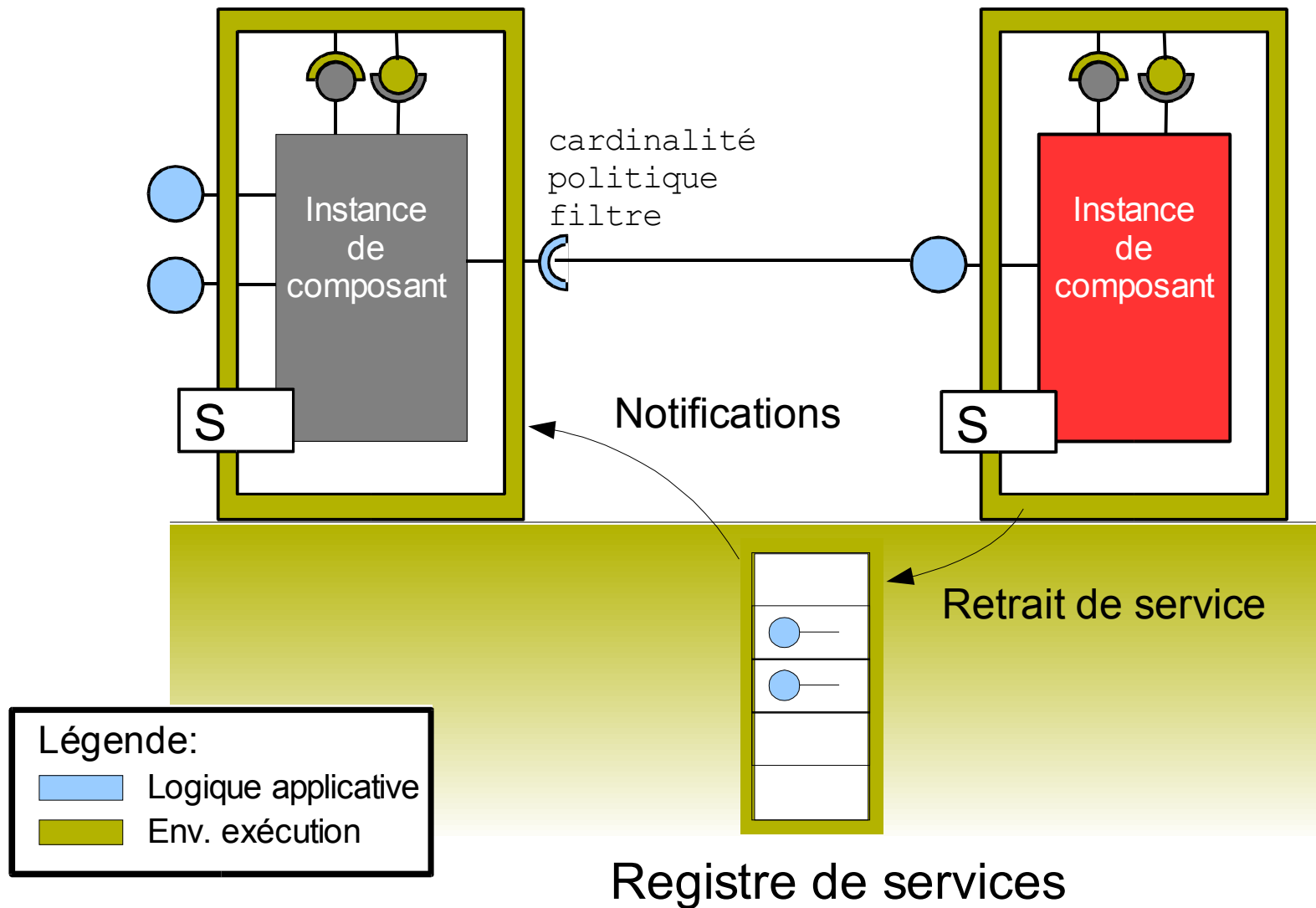


Gestion des instances



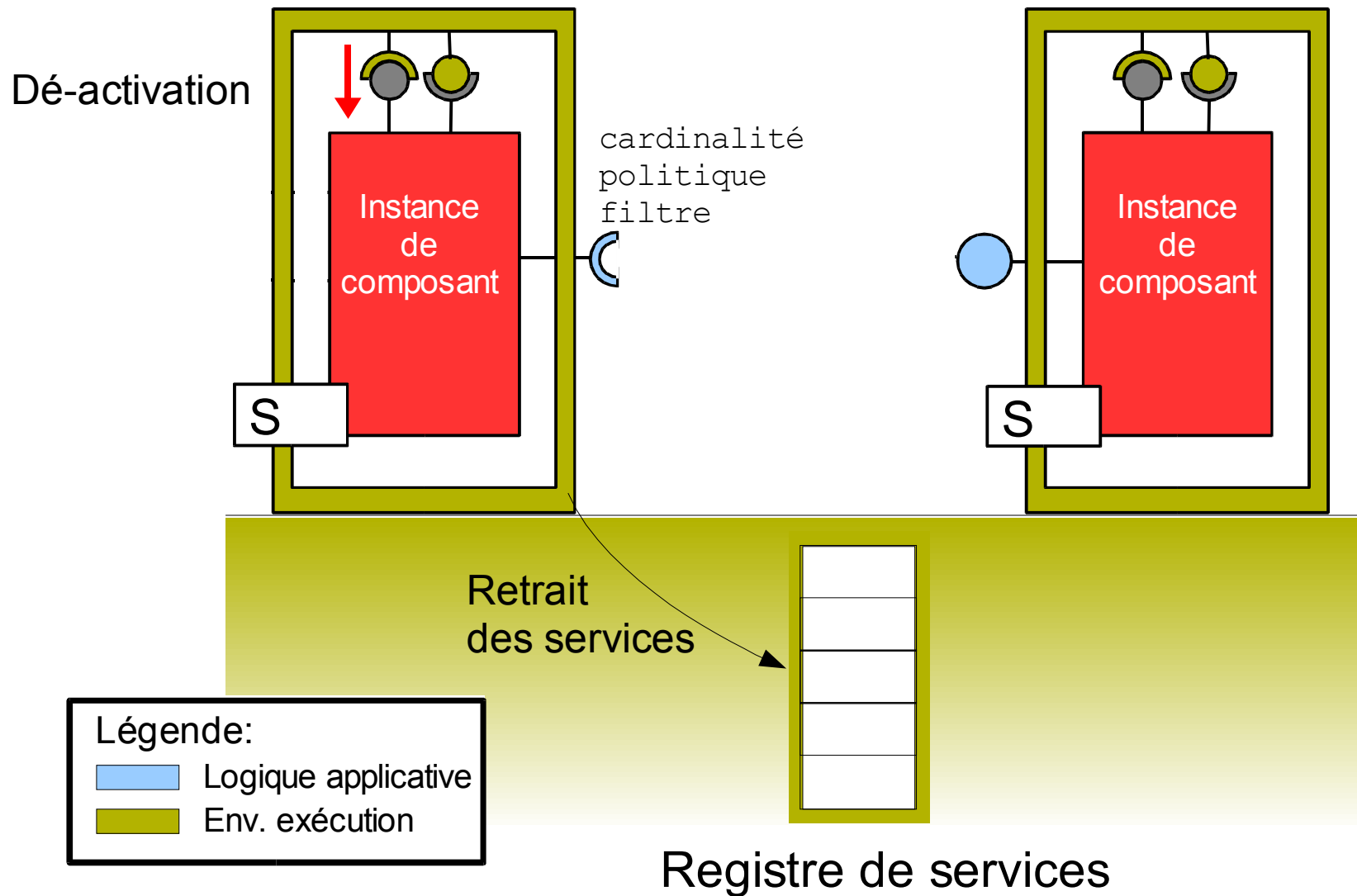


Gestion des instances





Gestion des instances






Plan de la thèse

Problématique

Fondations

Modèle à composants orienté services

- Composants à services
- Gestion des instances
-  ● Assemblage et évolution d'une application
- Types d'instances
- Imprévisibilité et espaces de résolution
- Gestion de composition
- Descripteur de composition

Réalisation

Évaluations

Conclusions et perspectives



Applications

Application

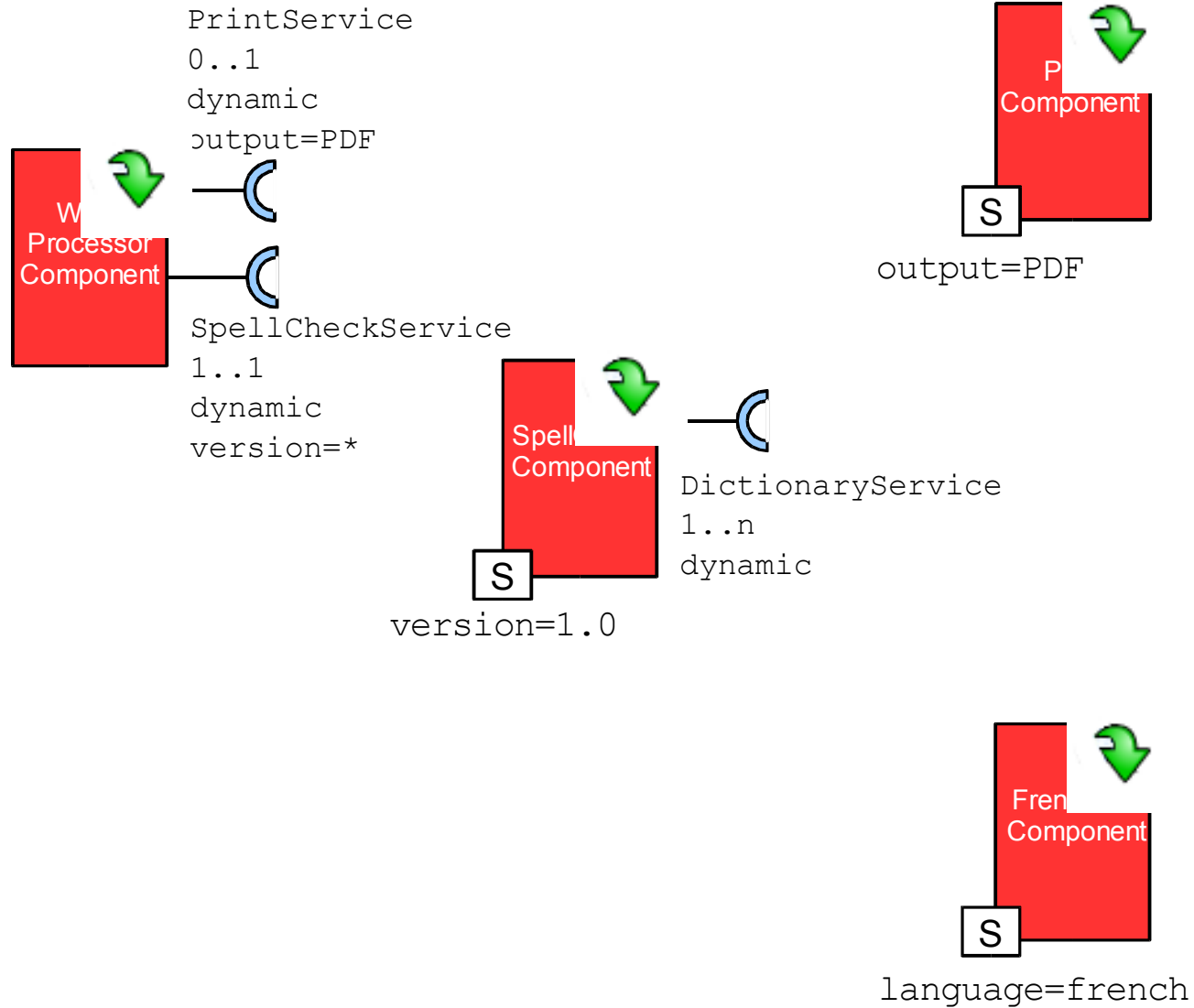
- Créée à partir d'ensemble d'instances introduites dans un environnement d'exécution
- Un "noyau" extensible

Gestion résulte dans applications capables

- D'auto-assemblage progressif
- D'incorporer des fonctionnalités
- De libérer des fonctionnalités
- De substituer des fonctionnalités

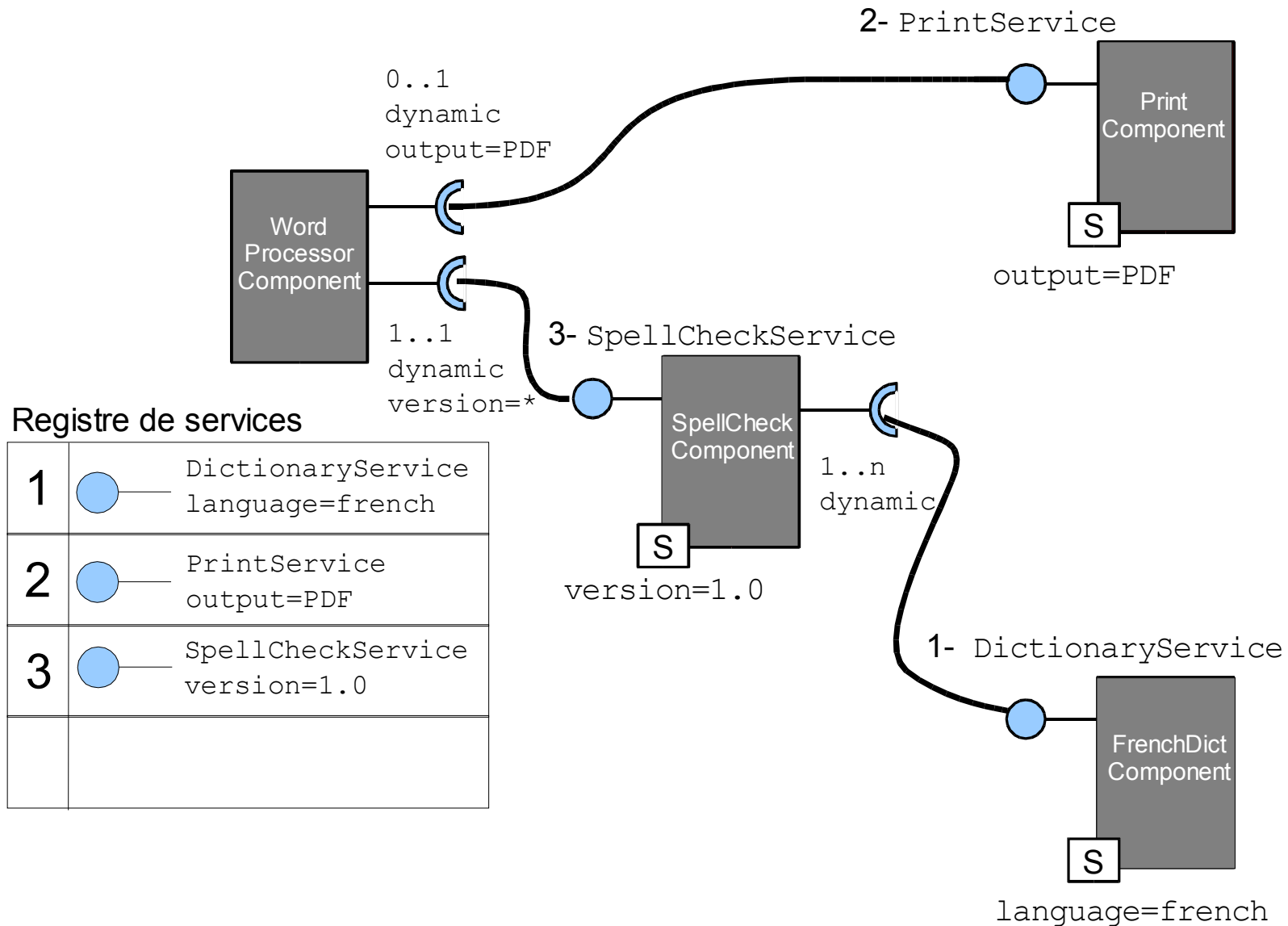


Assemblage



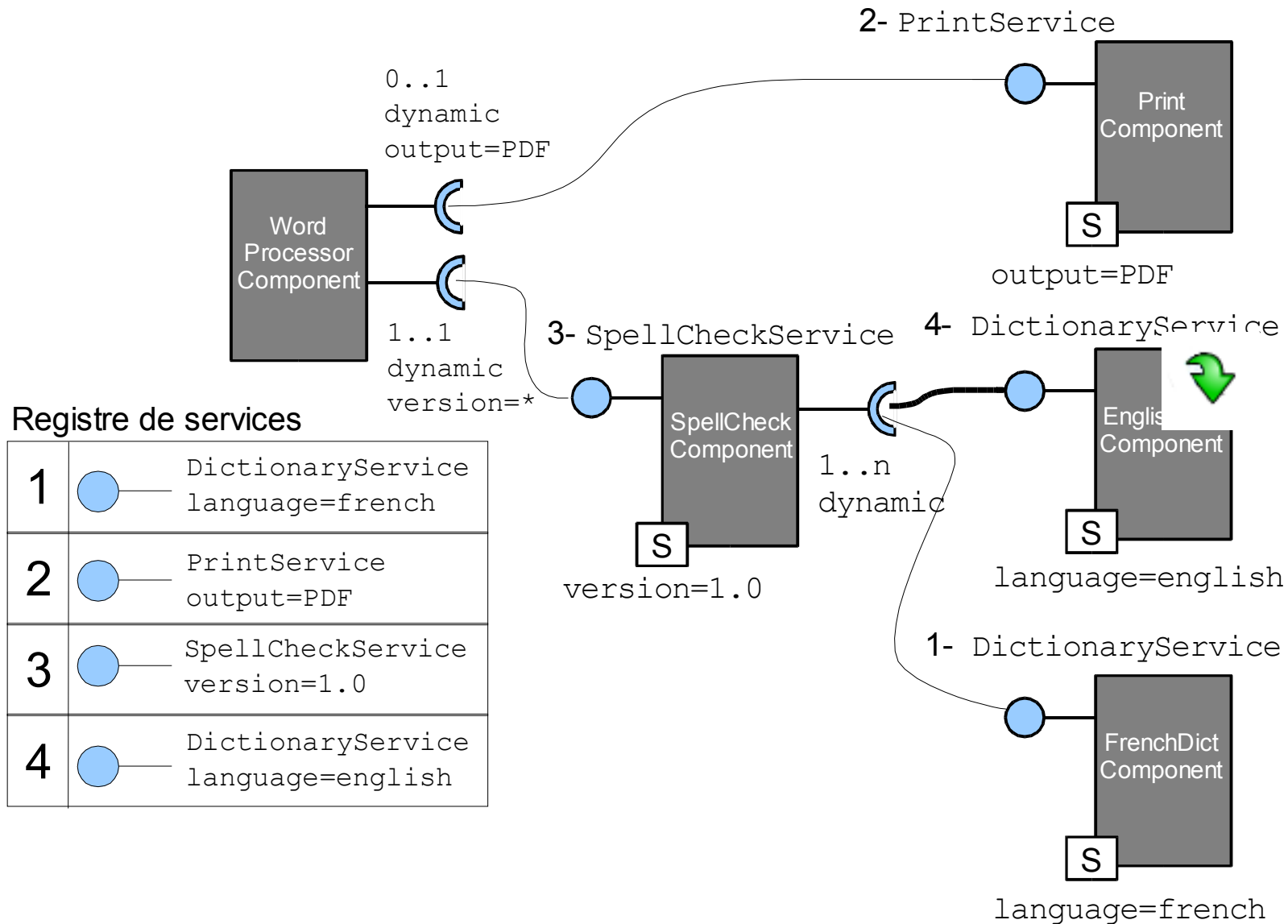


Assemblage



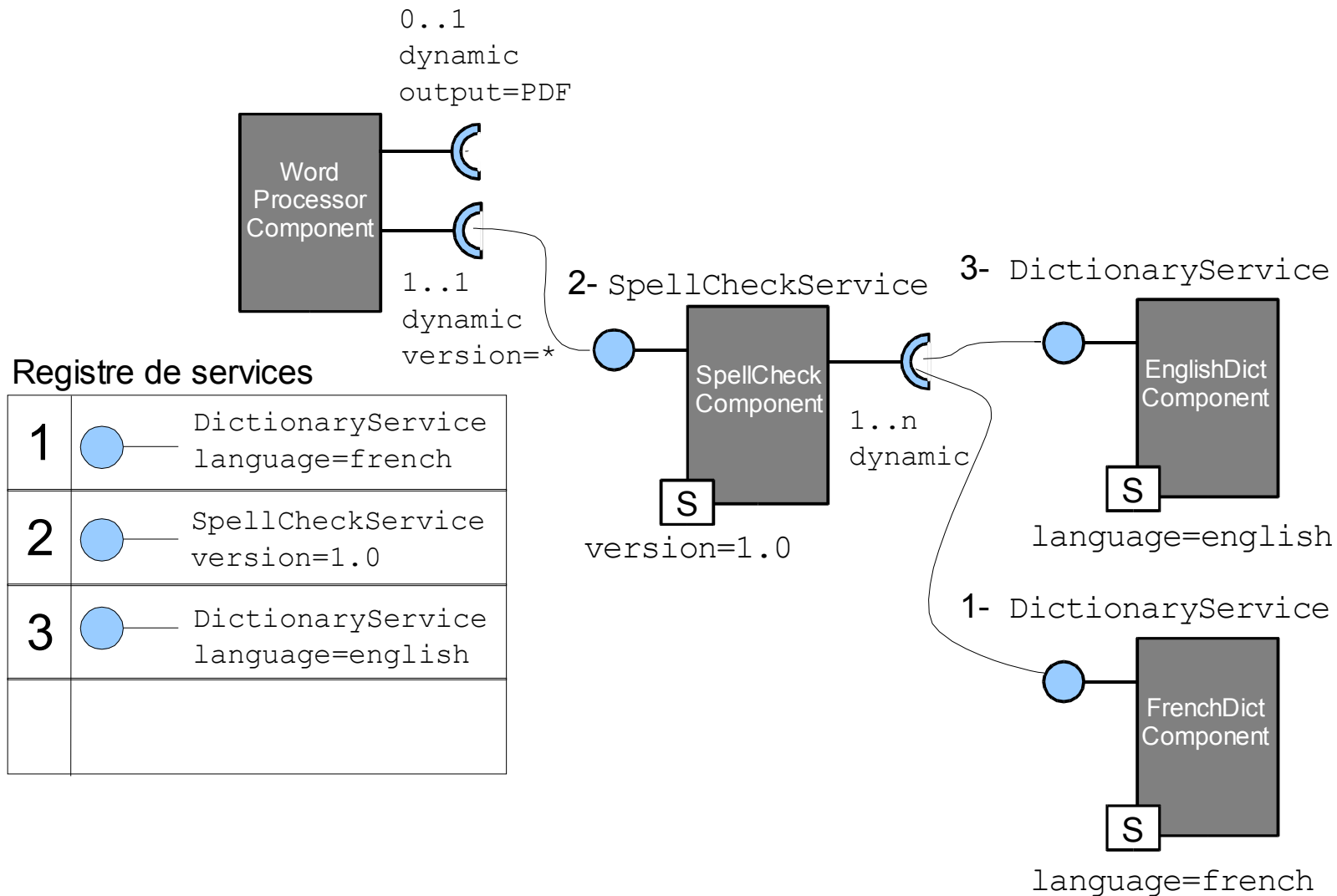


Incorporation de fonctionnalité



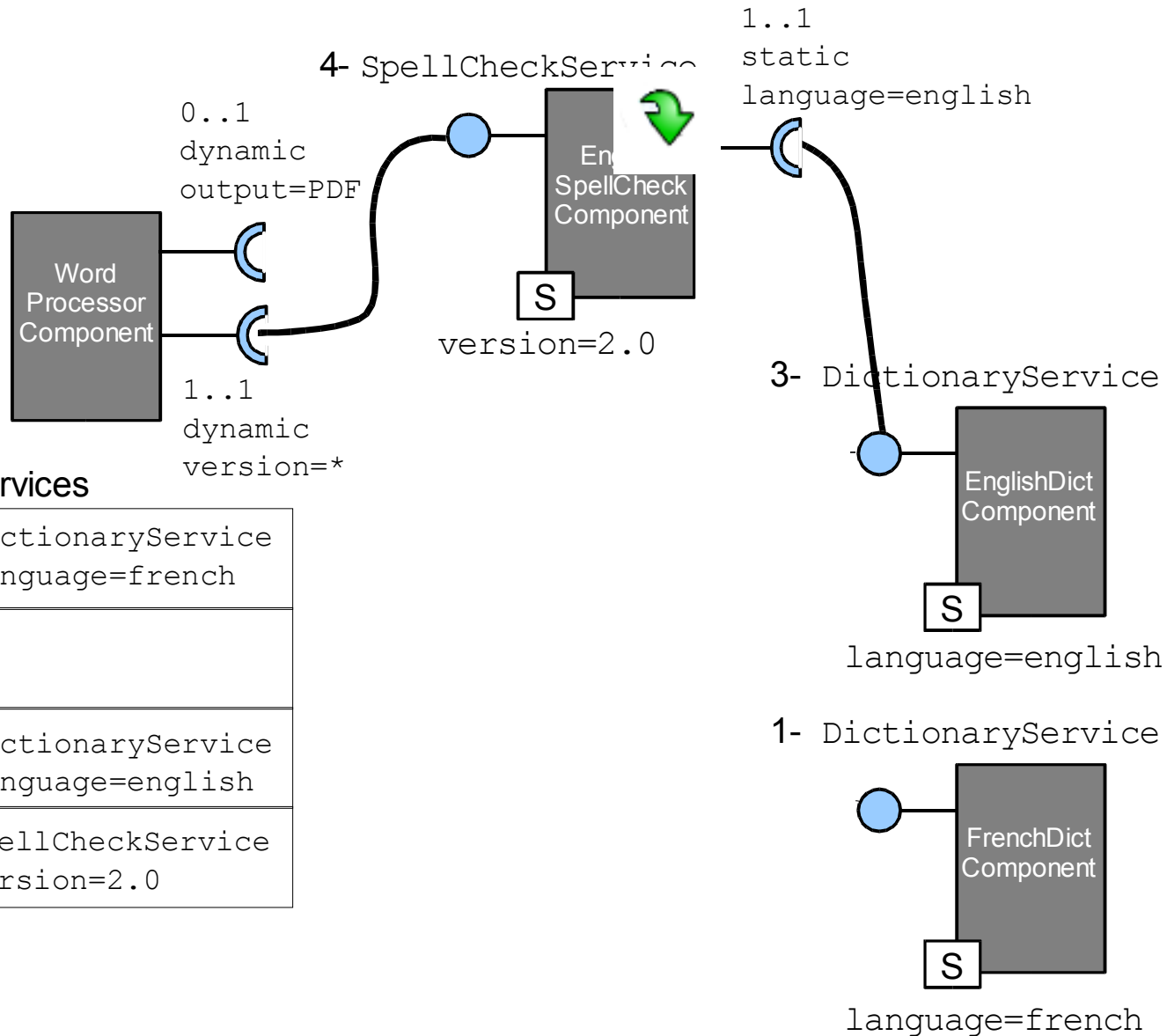


Départ de fonctionnalité





Substitution de fonctionnalité

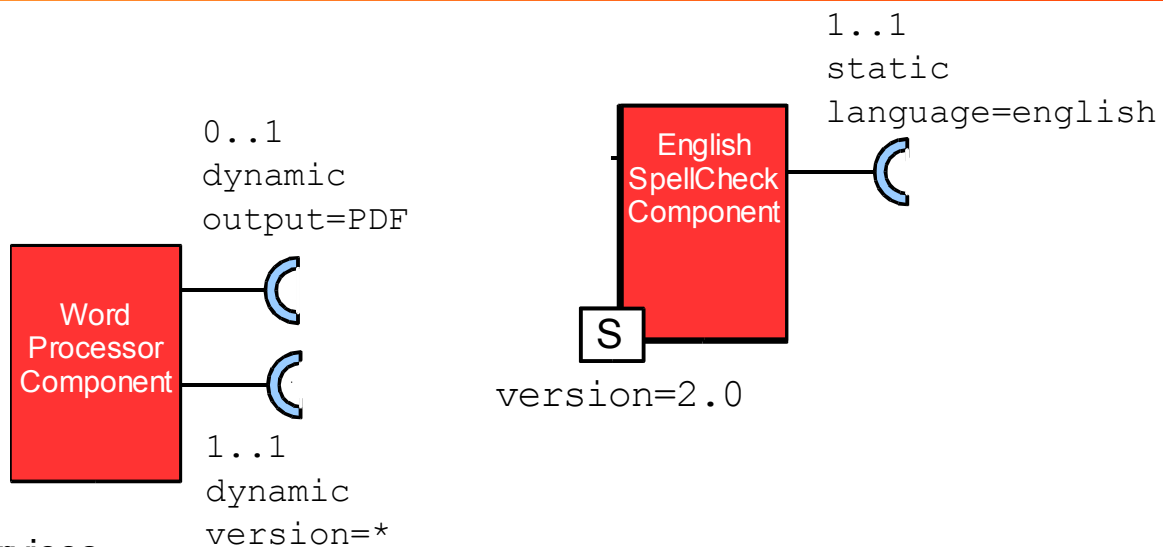


Registre de services


1	●	DictionaryService language=french
3	●	DictionaryService language=english
4	●	SpellCheckService version=2.0



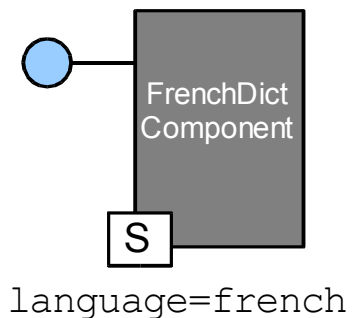
Invalidation « en chaine »



Registre de services

1	 DictionaryService language=french

1- DictionaryService






Plan de la thèse

Problématique

Fondations

Modèle à composants orienté services

- Composants à services
- Gestion des instances
- Assemblage et évolution d'une application
-  ● Types d'instances
- Imprévisibilité et espaces de résolution
- Gestion de composition
- Descripteur de composition

Réalisation

Évaluations

Conclusions et perspectives



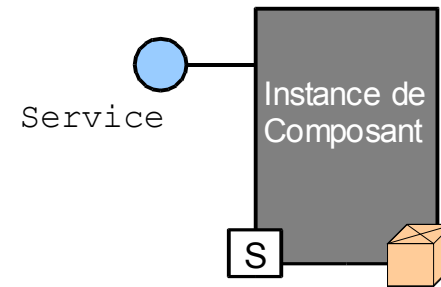
Types d'instances

Instances de déploiement


- Instances uniques (singleton)
- Cycle de vie lié aux activités de déploiement

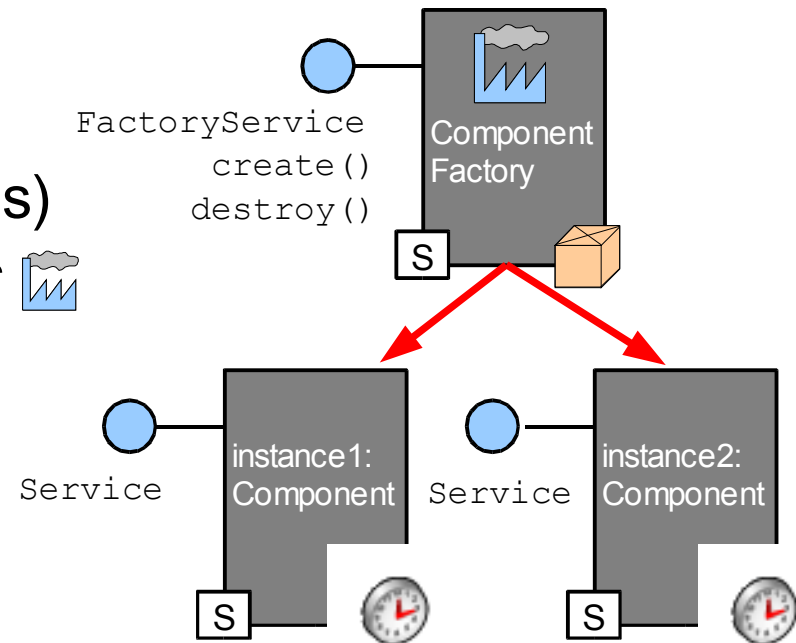
`create()` = installation

`destroy()` = retrait



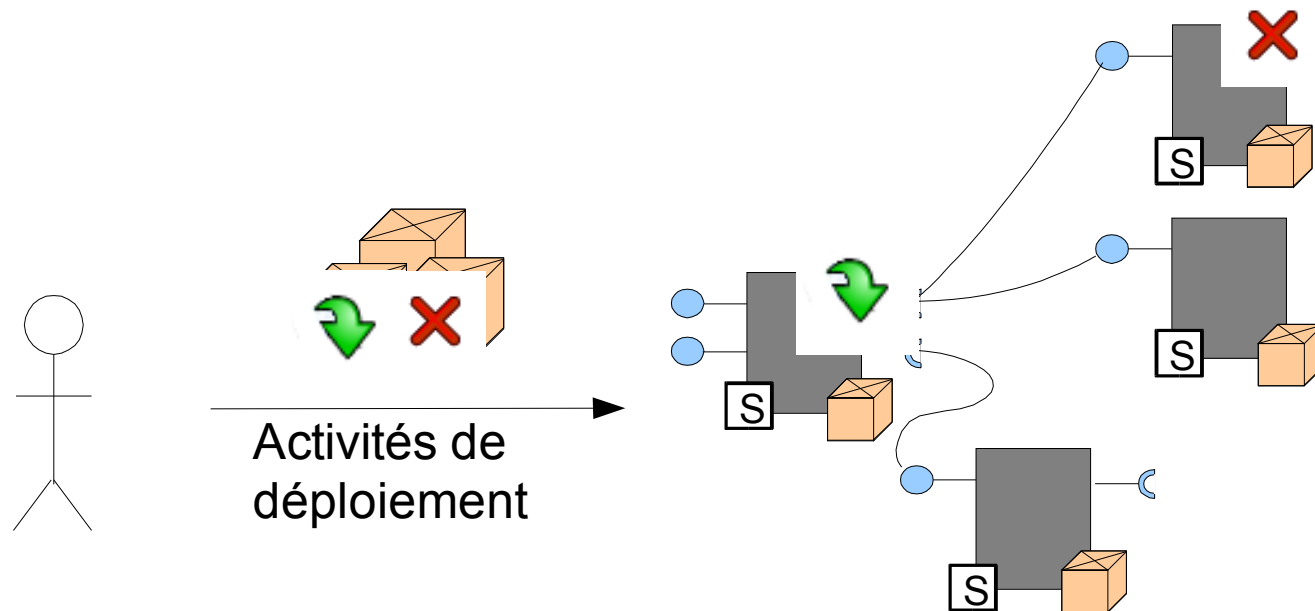
Instances dynamiques

- Instances multiples (nommées)
- Création à partir de *fabriques* 
 - Instances de déploiement
 - Disponibilité dynamique





Apps. avec instances de déploiement

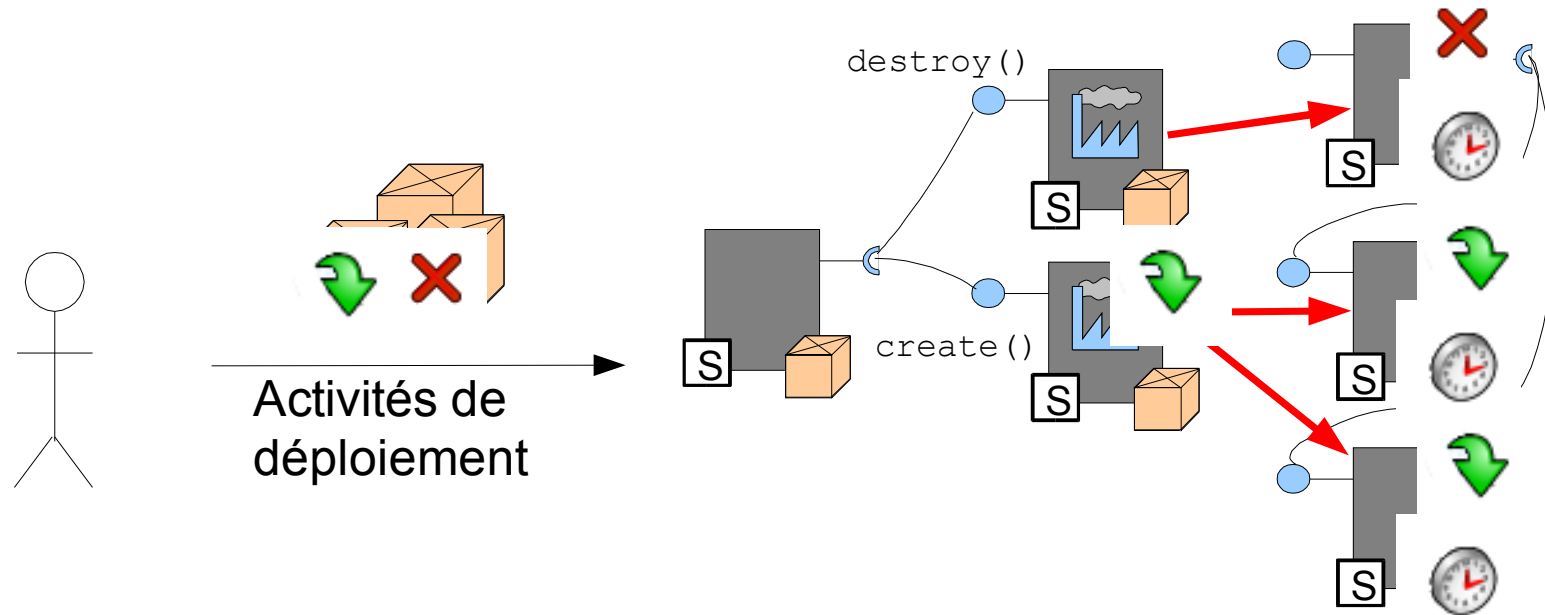


Assemblage et évolution d'une application

- Par rapport aux activités de déploiement



Applications avec instances dynamiques



Assemblage et évolution d'une application

- Par rapport aux activités de déploiement
- Par rapport aux interactions avec les fabriques à l'intérieur du système



Plan de la thèse

Problématique

Fondations

Modèle à composants orienté services

- Composants à services
- Gestion des instances
- Assemblage et évolution d'une application
- Types d'instances
- Imprévisibilité et espaces de résolution
- Gestion de composition
- Descripteur de composition

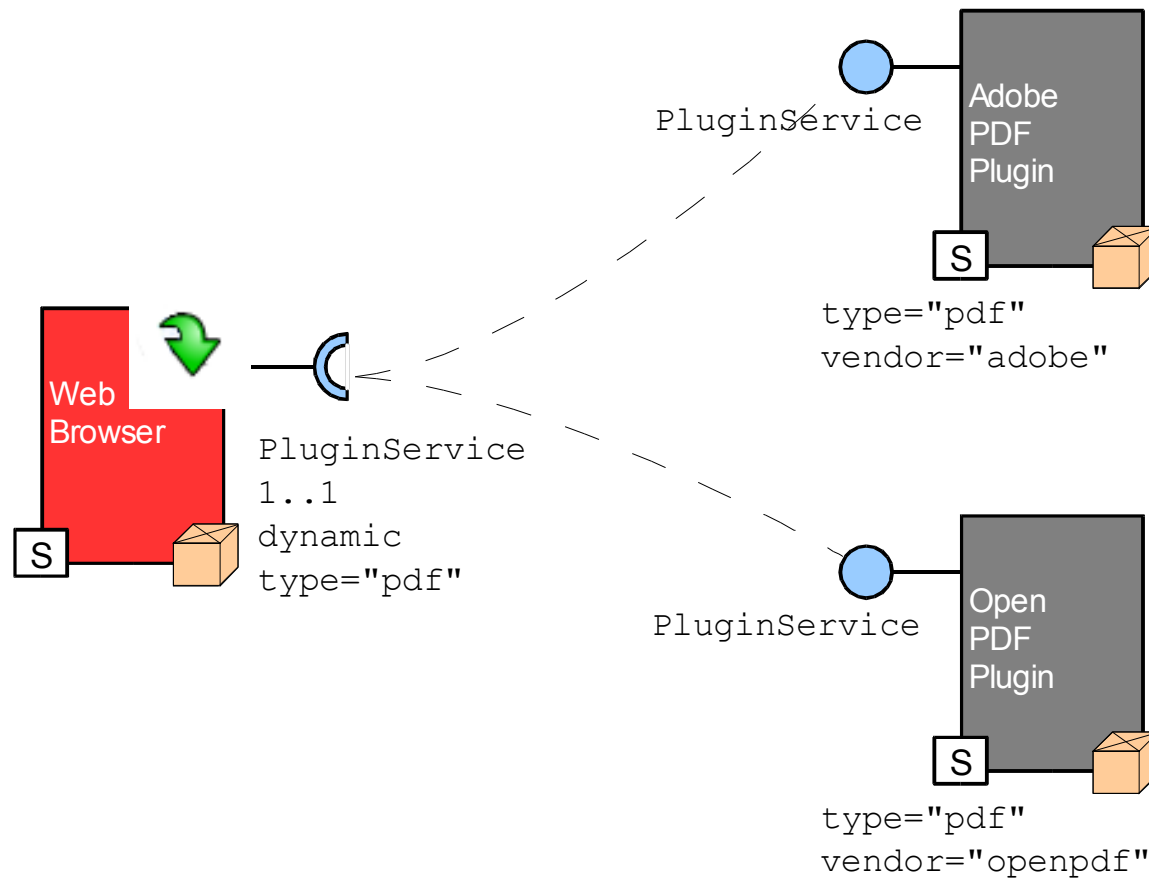
Réalisation

Évaluations

Conclusions et perspectives

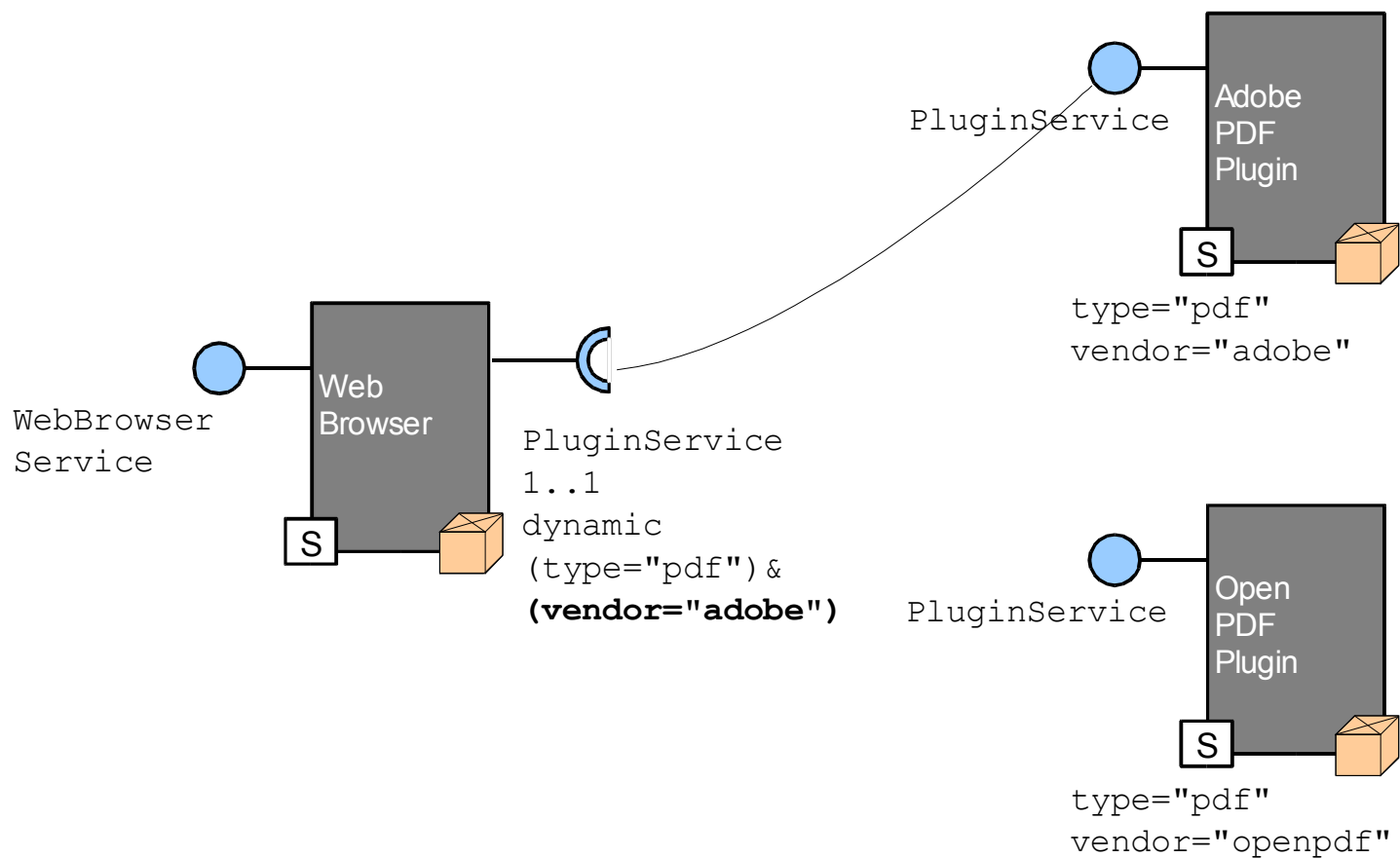


Imprévisibilité



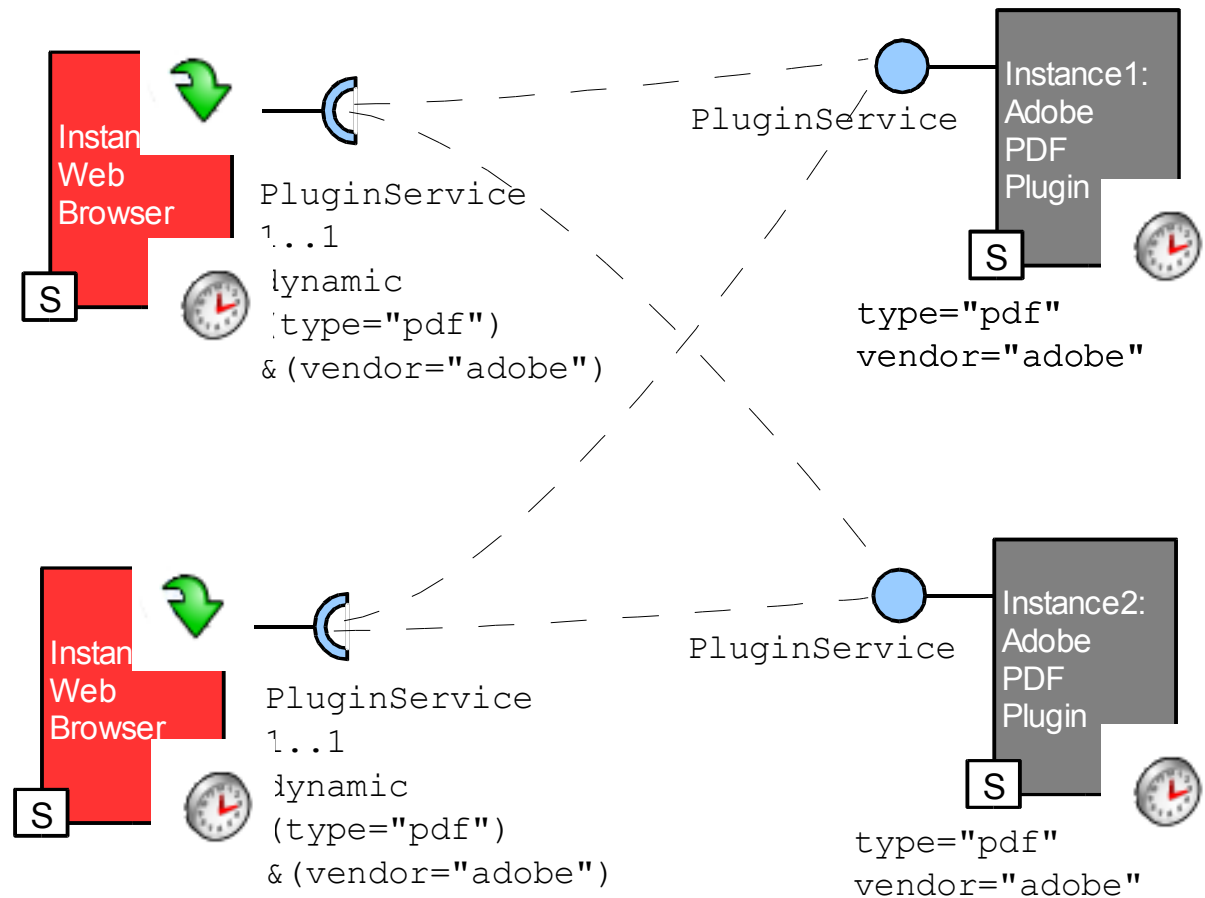


Imprévisibilité et instances déploiement



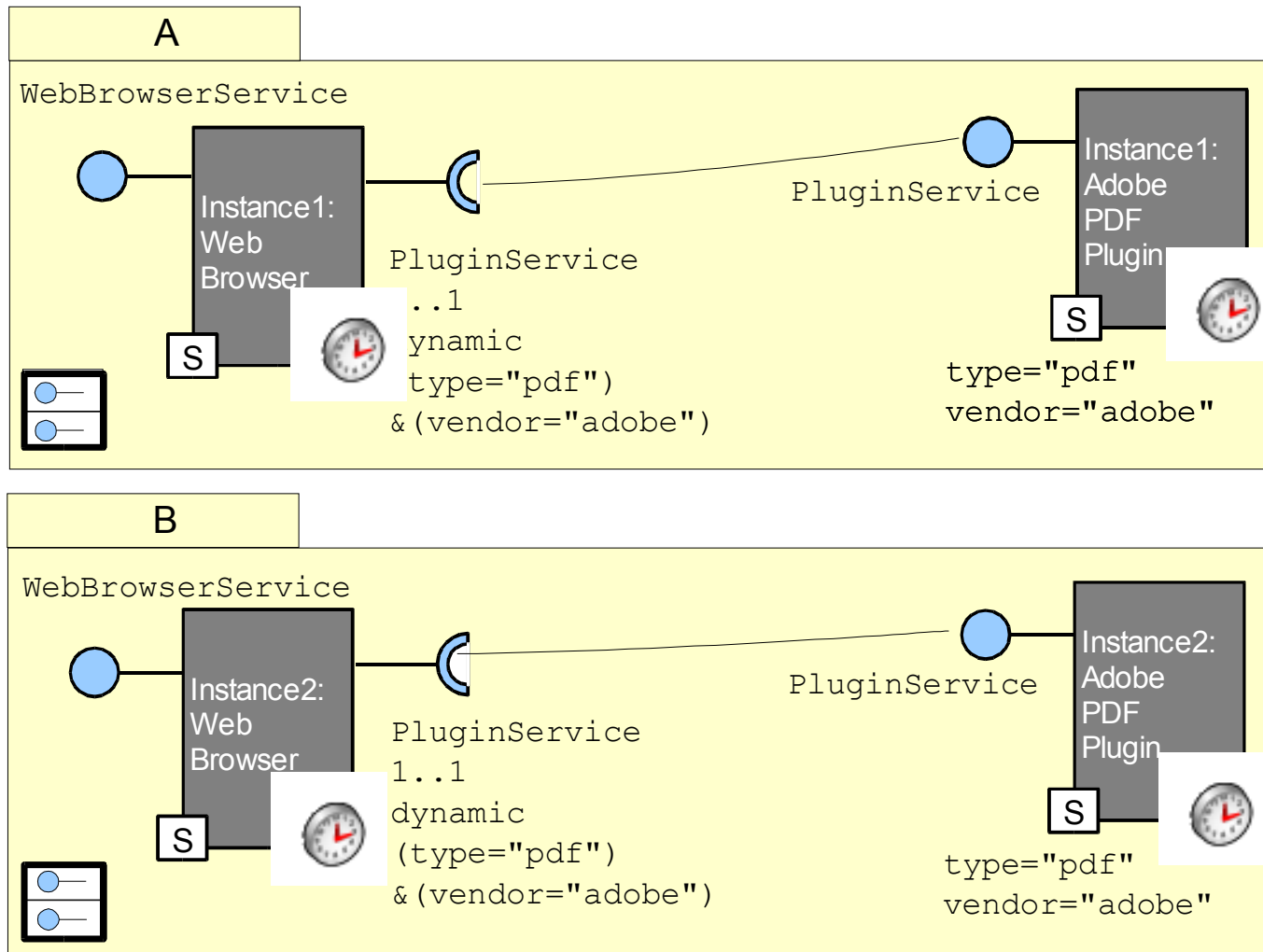


Imprévisibilité et instances dynamiques



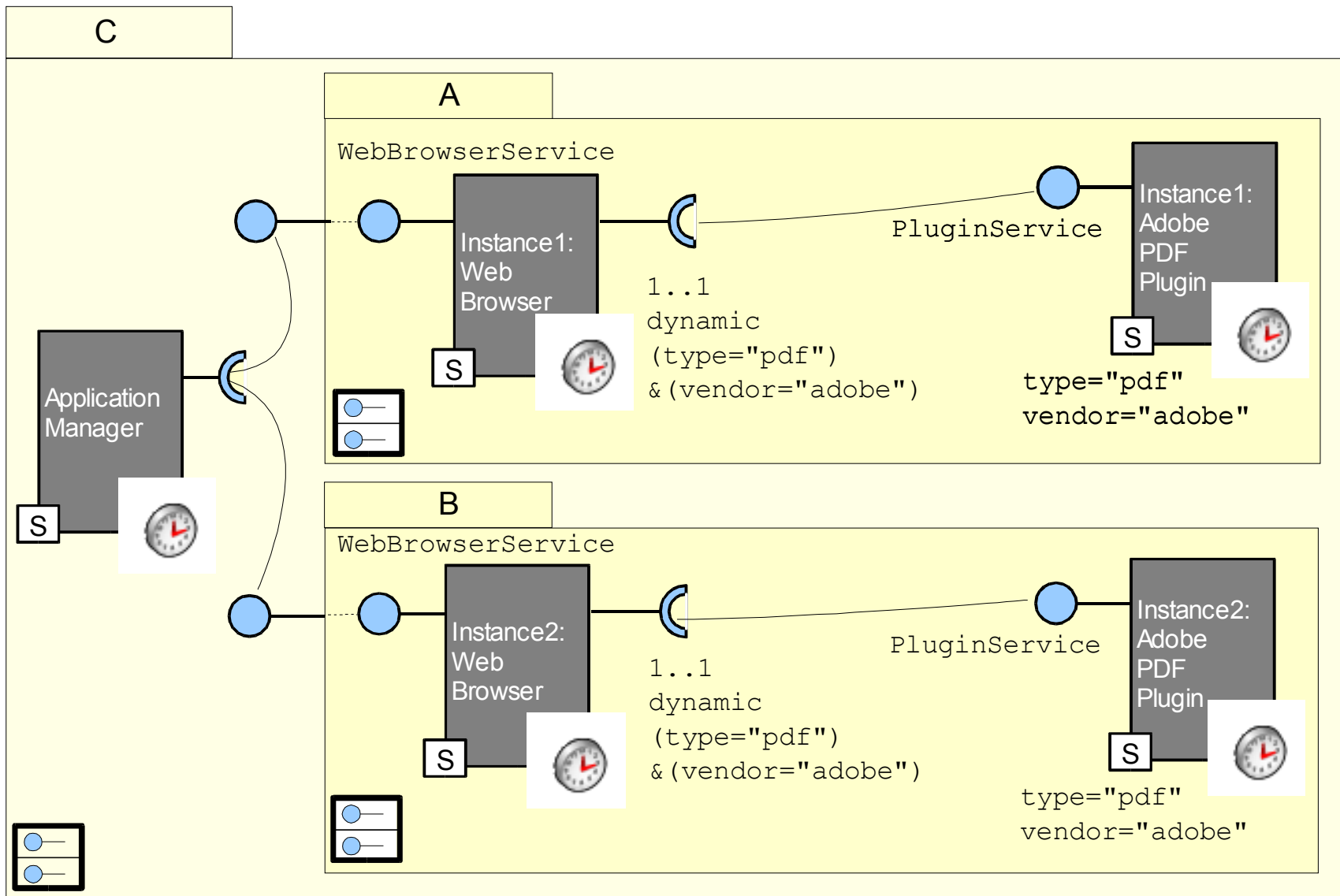


Espaces de résolution





Espaces de résolution





Plan de la thèse

Problématique

Fondations

Modèle à composants orienté services

- Composants à services
- Gestion des instances
- Assemblage et évolution d'une application
- Types d'instances
- Imprévisibilité et espaces de résolution
- Gestion de composition
- Descripteur de composition

Réalisation

Évaluations

Conclusions et perspectives



Composition d'instances

Composition

- Ensemble instances dans un espace de résolution
- Peut être hiérarchique

Création de composition

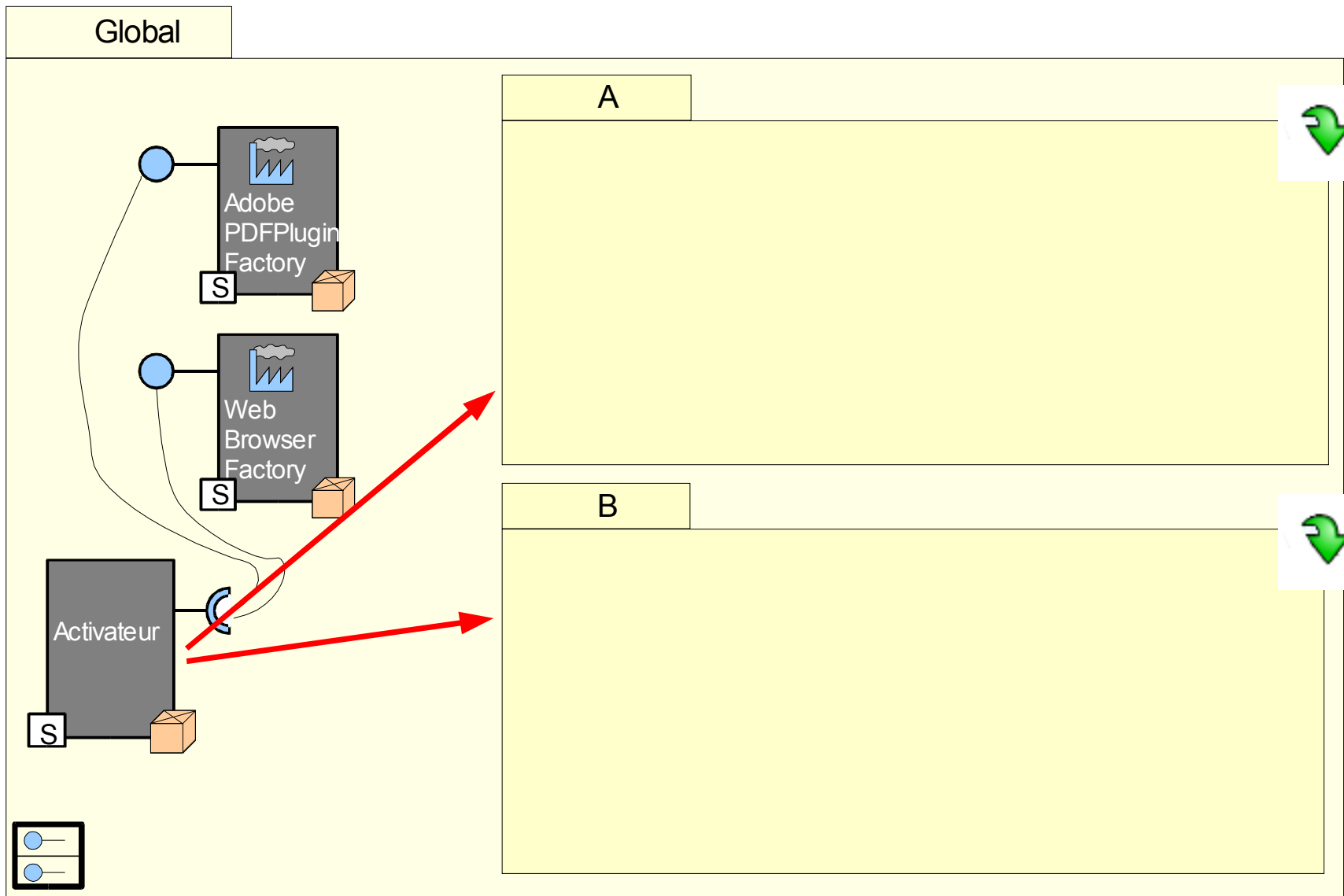
- Liaison avec fabriques
- Création d'espaces de résolution
- Création des instances

Gestion à l'exécution

- Arrivée et départ de fabriques

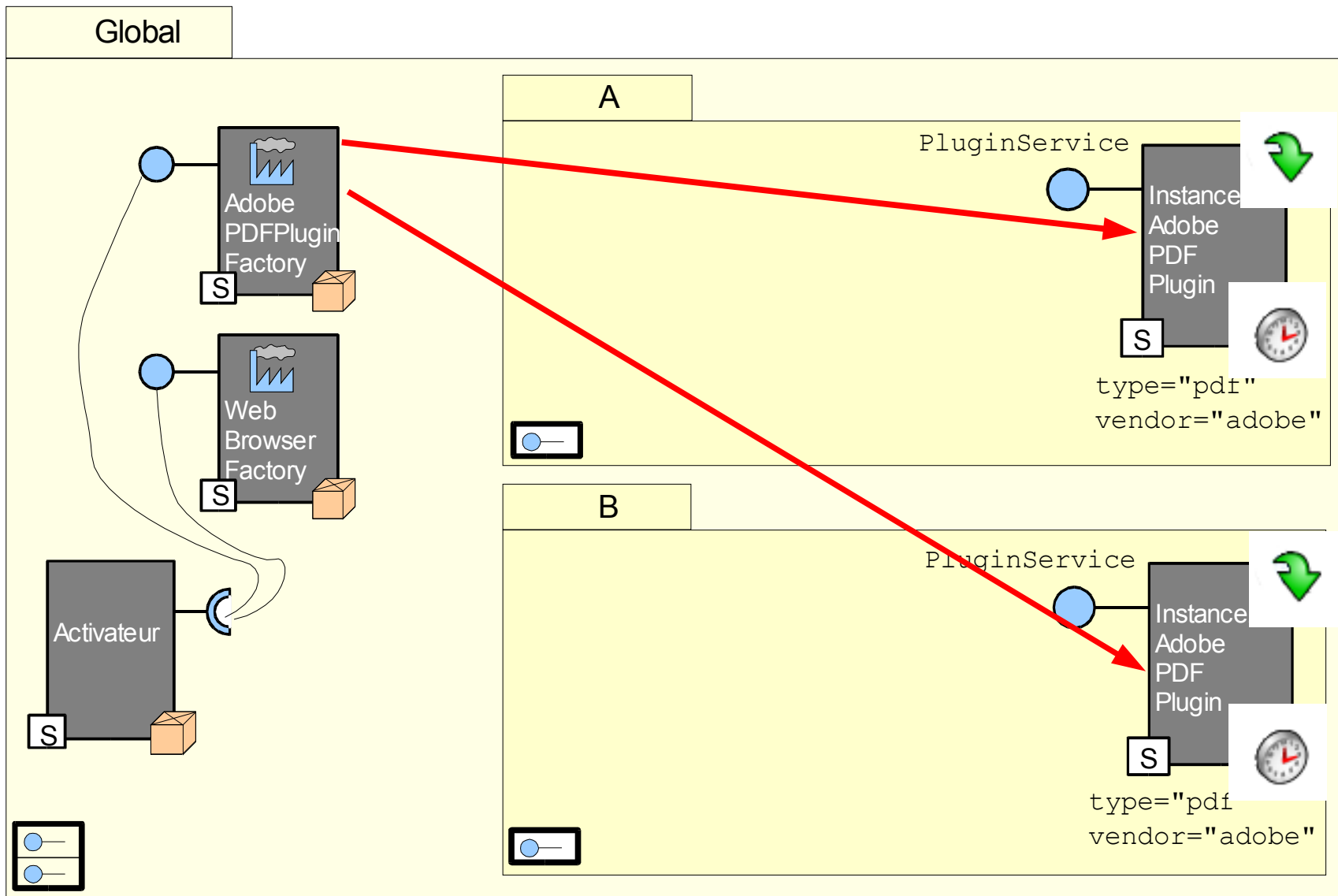


Création de composition



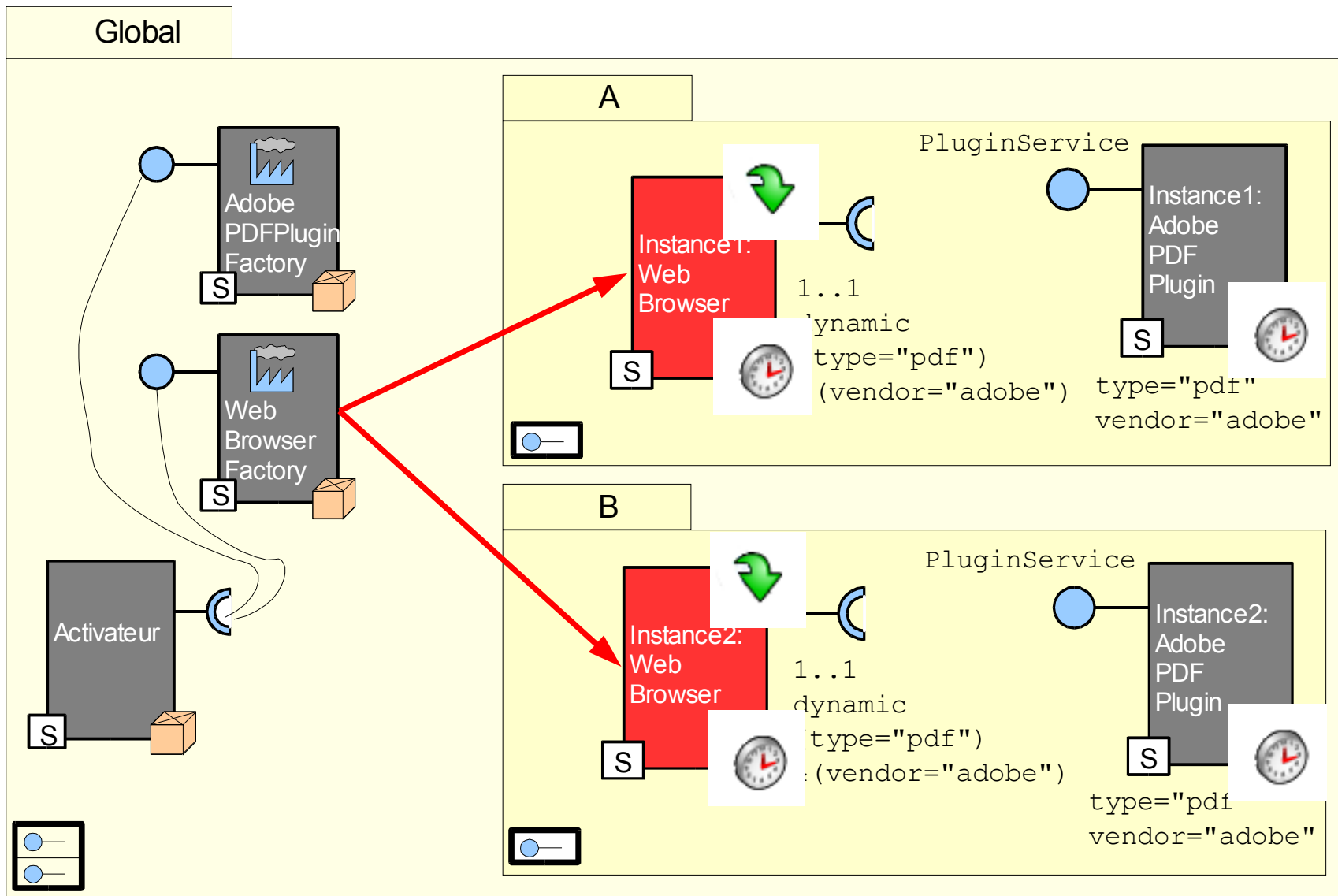


Création de composition



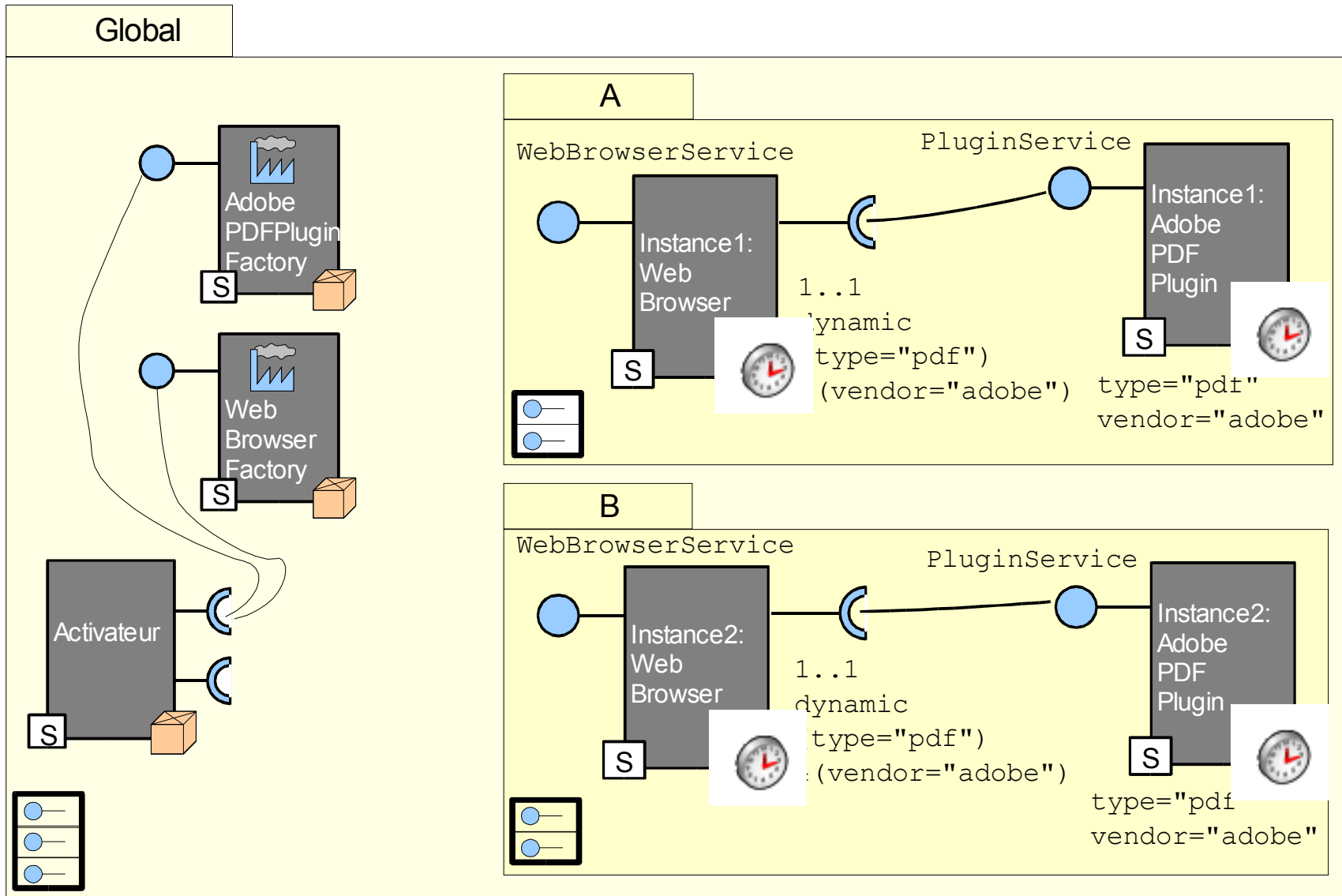


Création de composition



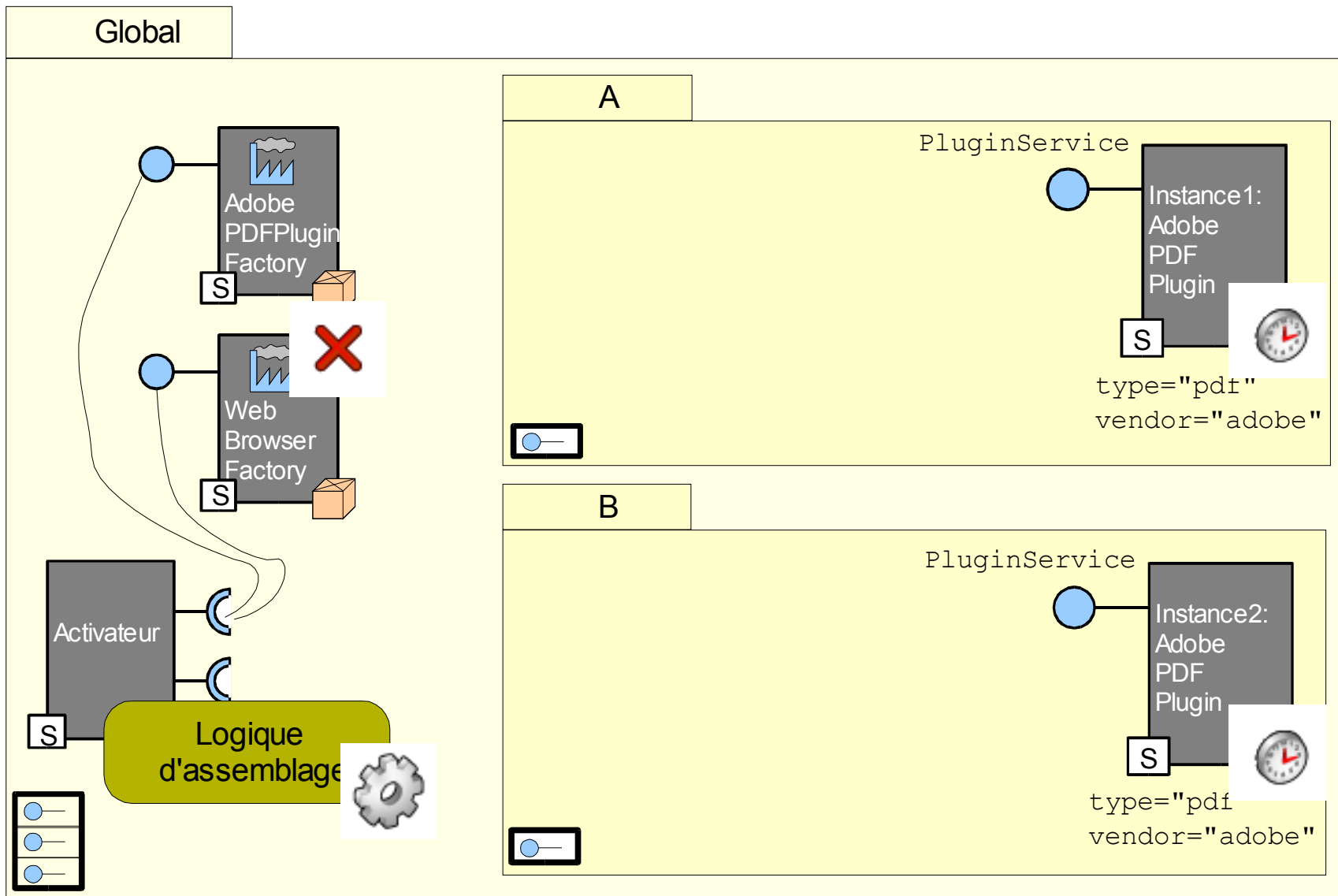


Création de composition





Création de composition





Plan de la thèse

Problématique

Fondations

Modèle à composants orienté services

- Composants à services
- Gestion des instances
- Assemblage et évolution d'une application
- Types d'instances
- Imprévisibilité et espaces de résolution
- Gestion de composition

 ● Descripteur de composition

Réalisation

Évaluations

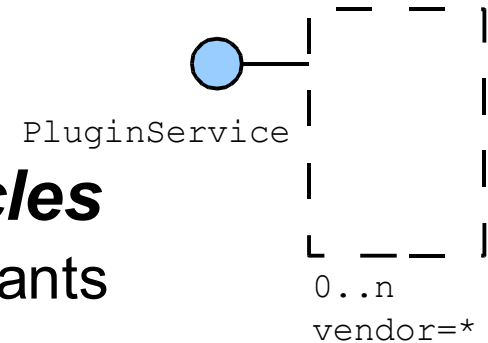
Conclusions et perspectives



Descripteur de composition

Décrit une composition de façon abstraite

- Extraire logique d'assemblage



Composition décrite en terme de socles

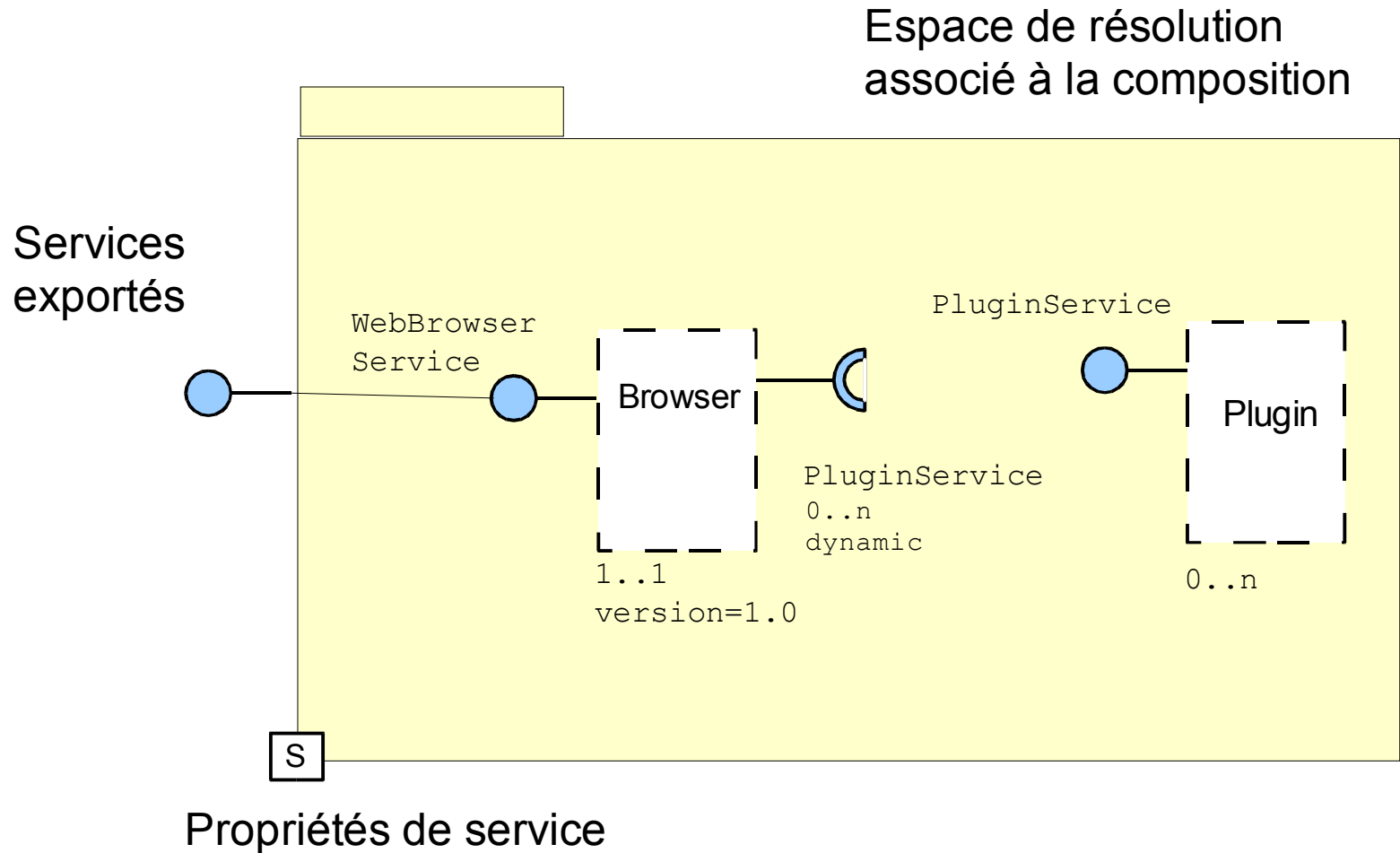
- Nombre variable d'instances de composants
- Vue externe d'un composant à services
- Connexions non-explicites

Intégration à l'exécution

- Informations sur vue externe associées aux fabriques
- Tout composant à service qui implémente vue externe est candidat à être intégré
- Services fournis additionnels permis
- Service requis additionnels permis si optionnels

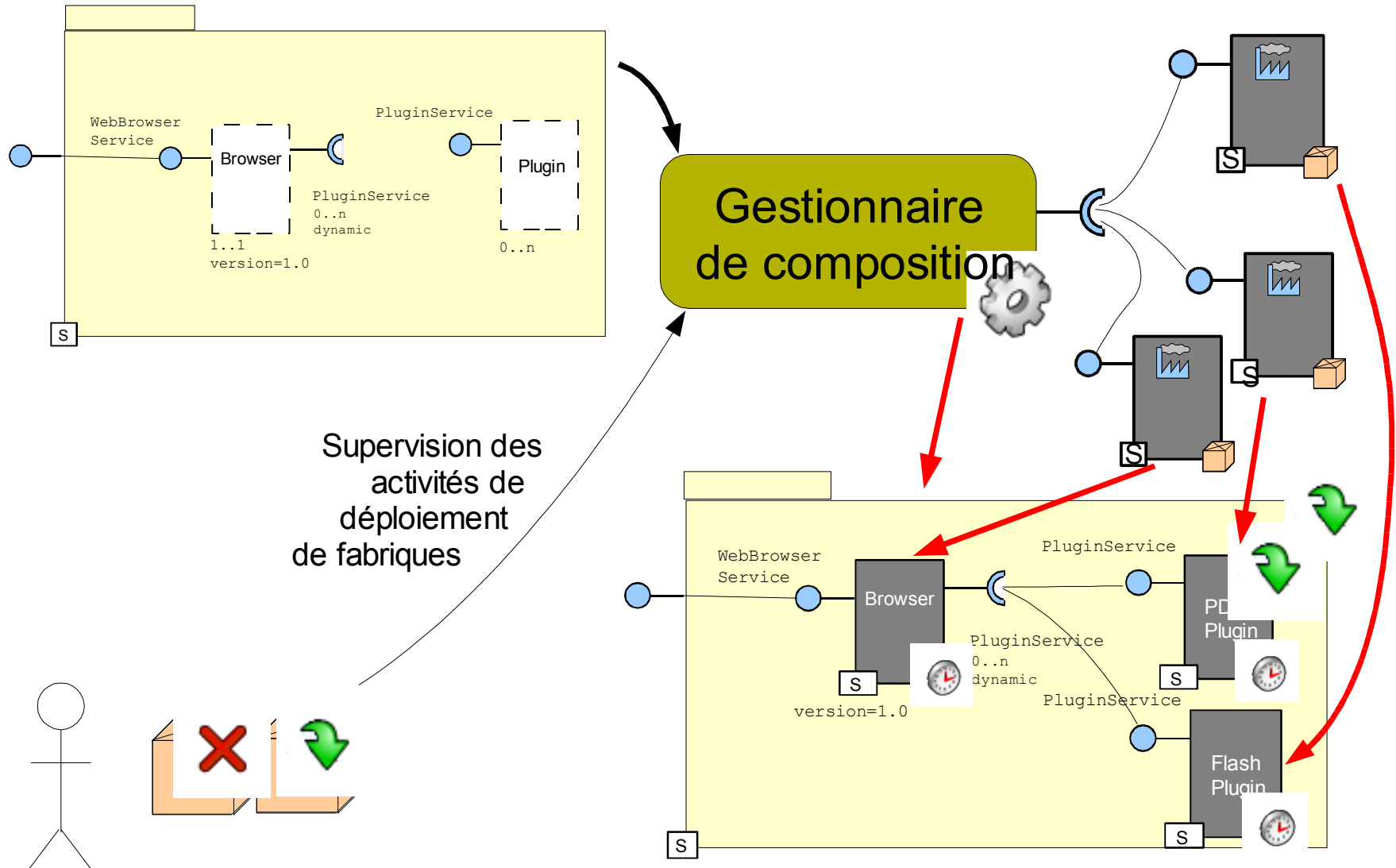


Descripteur de composition





Gestionnaire de composition





Plan de la thèse

Problématique

Fondations

Modèle à composants orienté services

Réalisation

- Service Binder
- Composition Manager
- Service d'introspection

Évaluations

Conclusions et perspectives



Contraintes

Description de services uniquement syntaxique

- Interfaces Java

Pas de gestion du transfert d'état

- Lors de substitutions
- Pas de traitement des propriétés de configuration

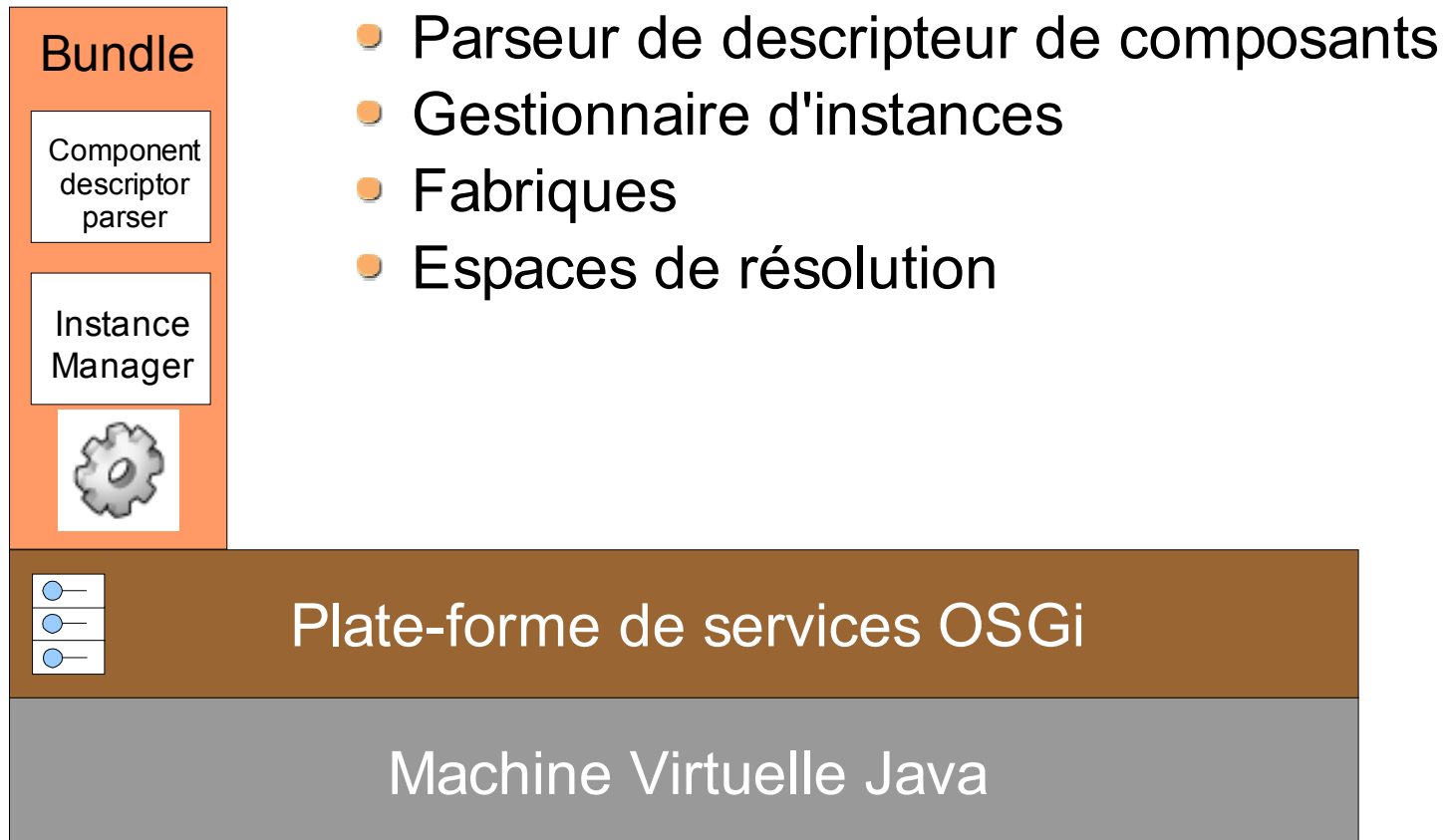
Réalisation au dessus de OSGi

- Registre de services avec mécanismes d'interrogation (LDAP)
- Mécanismes de notifications
- Gestion des aspects liés au déploiement (bundle)
- OSCAR



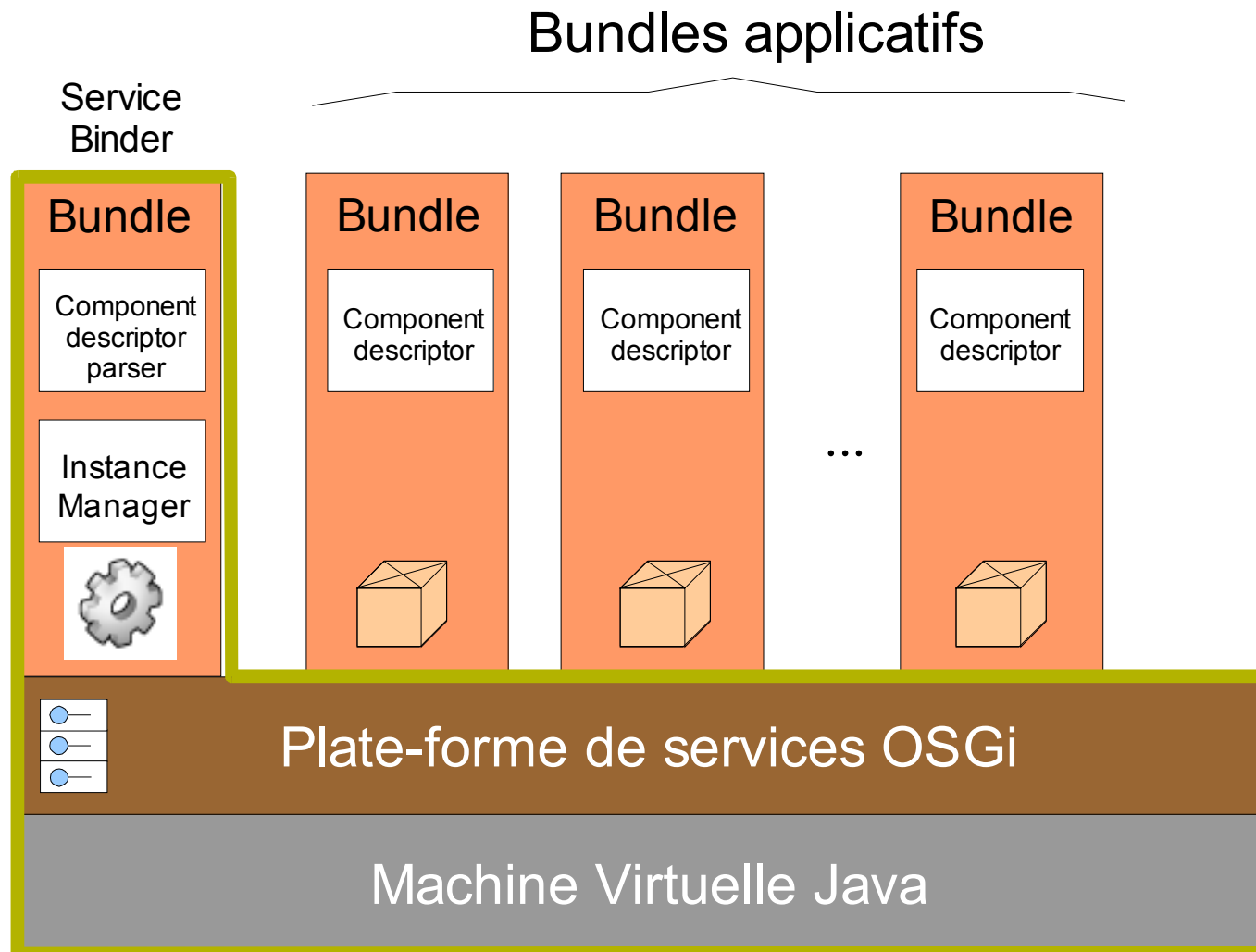
Service Binder

Service Binder





Service Binder





Descripteur de composant

● Descripteur composant

```
<bundle>
```

```
  <component class="org.examples.impl.SpellCheckComponent" factory="no">
```

```
    <provides service="org.examples.interfaces.SpellCheckService"/>
```

```
    <property name="version" value="1.0" type="string"/>
```

```
    <requires
```

```
      service="org.examples.interfaces.DictionaryService"
```

```
      filter="(language=*)" "
```

```
      cardinality="1..n"
```

```
      policy="dynamic"
```

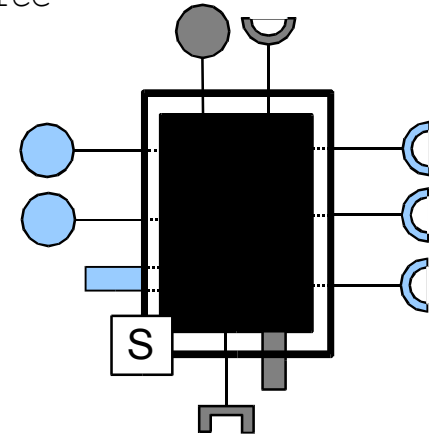
```
      bind-method="addDictionary"
```

```
      unbind-method="removeDictionary"
```

```
    />
```

```
  </component>
```

```
</bundle>
```



● Extension du Manifest

```
Bundle-Activator: org.examples.impl.Activator
```

```
Import-Package:
```

```
  org.ungoverned.gravity.servicebinder; specification-version="1.1",
```

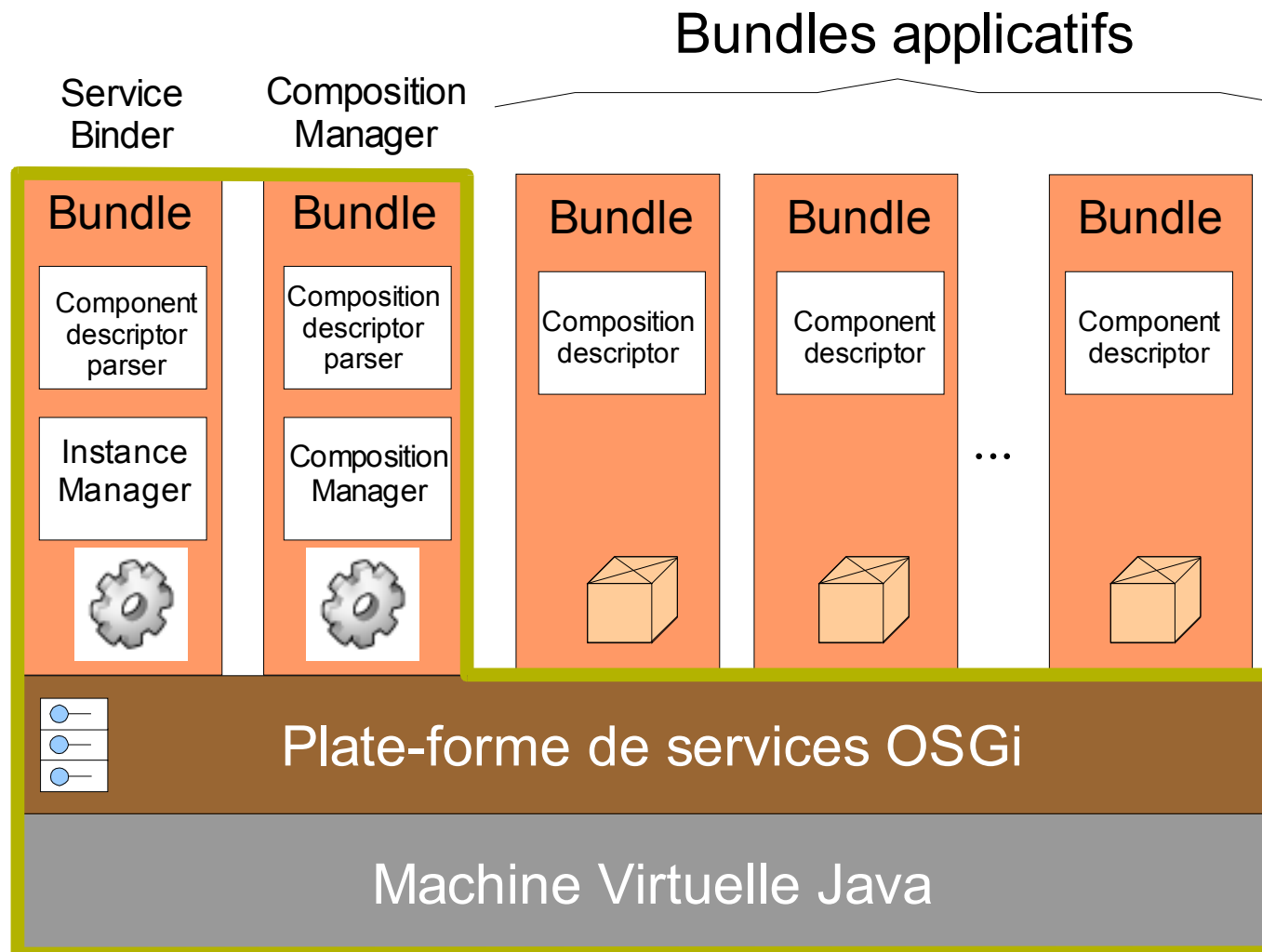
```
Bundle-Classpath: ., somelib.jar
```

```
Bundle-Version: 1.0.0
```

```
Metadata-Location: org/examples/impl/res/metadata.xml
```



Composition manager





Descripteur de composition

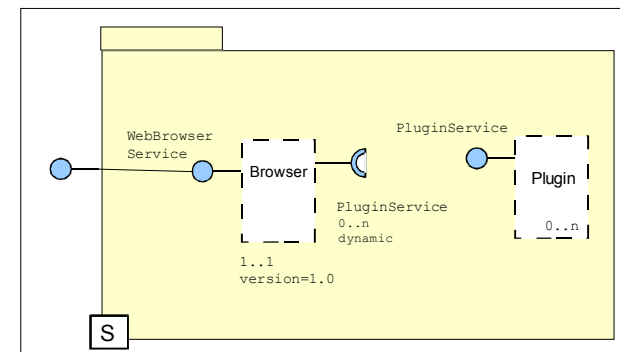
```
<composition factory="yes">
  <exports from="core" service="org.gravity.services.Application" type="provided"/>
  <exports from="core" service="org.gravity.services.WindowManager" type="required"/>

  <property name="appName" value="WebBrowser" type="string"/>

  <placeholder id="core" filter="" cardinality="1..1">
    <provides service="org.gravity.services.Application"/>
    <provides service="org.gravity.services.WebBrowser"/>
    <requires service="org.gravity.services.WindowManager"
      filter=""
      cardinality="1..1"
      policy="dynamic"/>
    <requires service="org.gravity.services.BrowserPlugin"
      filter=""
      cardinality="0..n"
      policy="dynamic"/>
  </placeholder>

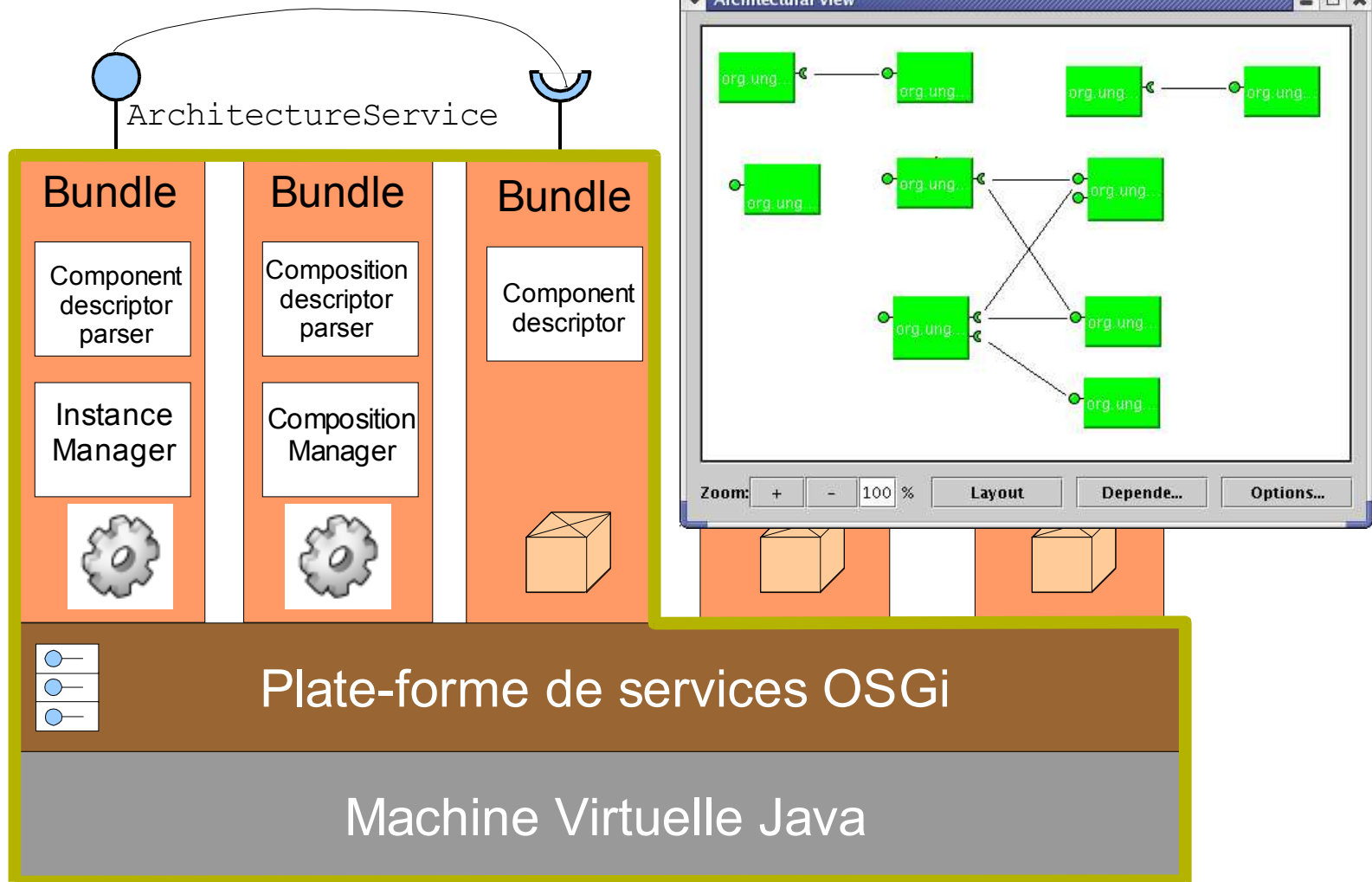
  <placeholder id="plug" filter=""
    cardinality="0..n">
    <provides
      service="org.gravity.services.BrowserPlugin"/>
  </placeholder>

</composition>
```





Service d'introspection





Plan de la thèse

Problématique

Fondations

Modèle à composants orienté services

Evaluations

- Service Binder et OSGi
- Schneider Electric
- Ascert
- Lanceur d'applications
- Gravity

Conclusions et perspectives



Service Binder et OSGi

Simplifie la construction d'applications OSGi

- Concepts additionnels compatibles
- Taille réduite (70k), surcout raisonnable:

Task	Standard OSGi (in bytes)	Service Binder (in bytes)	Delta
Registering one service	156	537	244%
Registering one service and requesting one service	864	1344	55%
Additional requested services	768	1032	34%

Comité d'OSGi intéressé par ces idées

- Incorporation dans future spécification ?

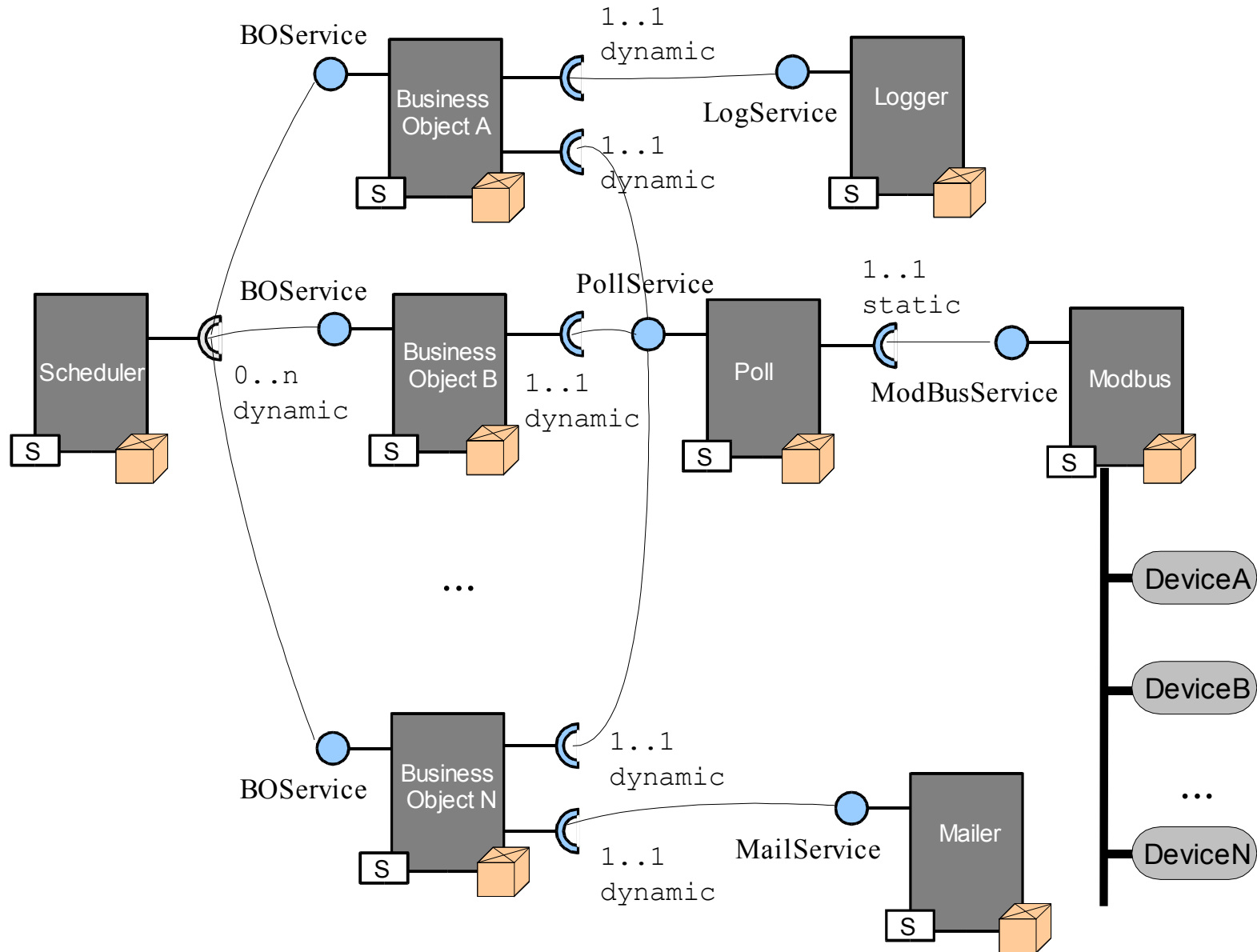


Projet source libre

<http://gravity.sf.net/servicebinder>

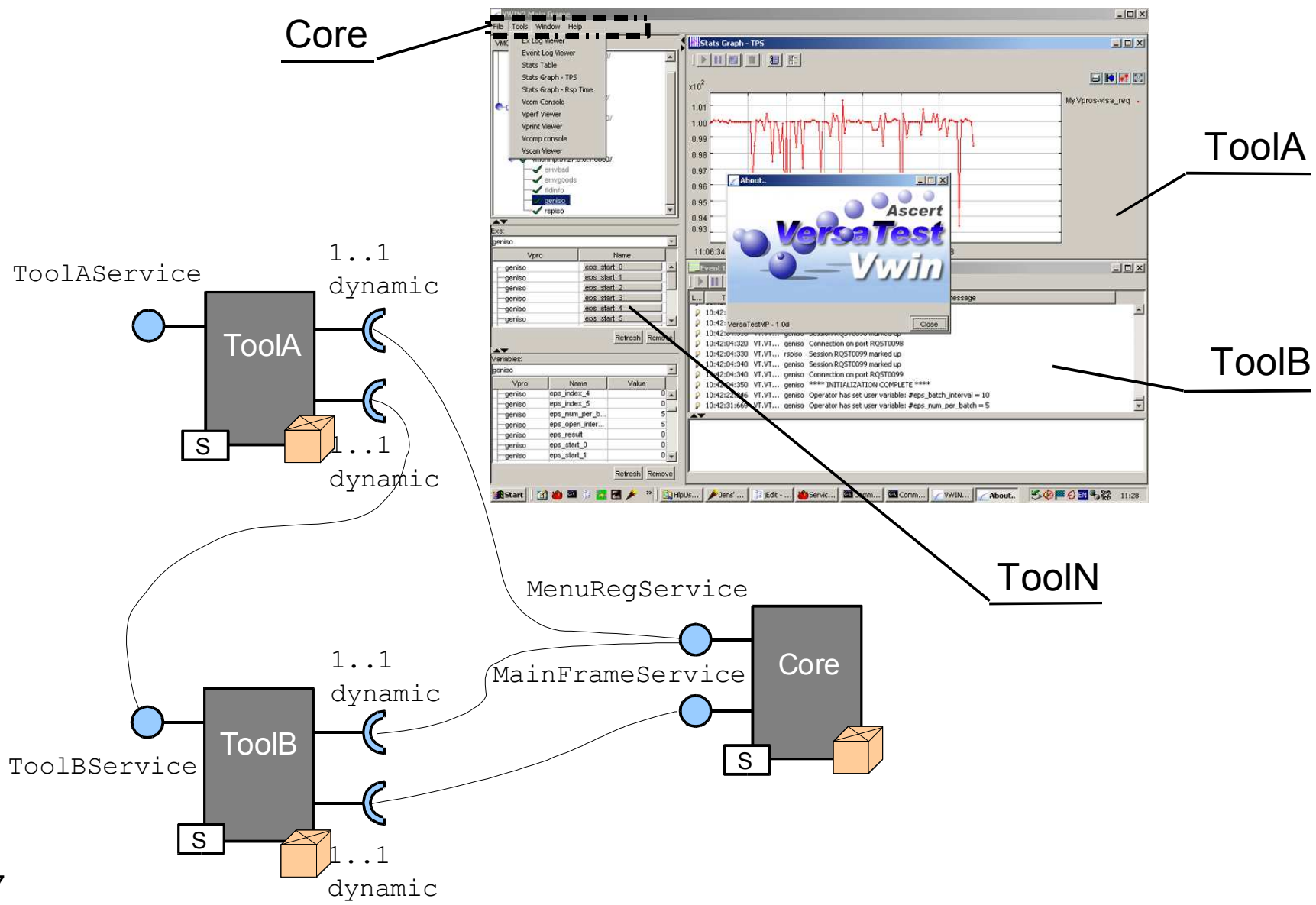


Schneider Electric





Client VersaTest





Gestionnaire de composition

ServiceBinder
Simplifying application development
Authors: [Humberto Cervantes](#), [Richard](#)

Summary

The ServiceBinder is a mechanism that automates **service dependency management**.

One of the most complex tasks faced in this platform is writing assembly and adaptation logic in a service-oriented environment. Inside such an environment, services exhibit **dynamic availability** (i.e., they arrive or depart at any time during execution), and applications must be capable of handling these situations. A problem faced by developers is that they are responsible for writing both application and adaptation logic. Adaptation logic is, in general, complex, as it requires monitoring and reconfiguration to be realized. Moreover, these two types of logic end up intermixed inside the code, making modifications more difficult.

The ServiceBinder solves this problem by extracting assembly and adaptation logic from the bundles and moving it into an execution environment that is deployed inside the framework as a standard bundle. Assembly and adaptation logic is configured by information contained in an [XML descriptor](#) that extends the bundle manifest. Applications built with the ServiceBinder are assembled dynamically and are capable of adapting themselves autonomously, for example by substituting a departing service, or by integrating new services that arrive as the application is being executed.

The ServiceBinder is small (~70k) and independent of any OSGi framework implementation. It has been used in companies such as [Schneider Electric](#) and [Ascort](#) to develop research and commercial products, respectively.

Links

- [Concepts](#): The concepts behind the ServiceBinder are explained.
- [Dynamically extensible applications](#): Discusses how plug-in applications based on components can be built.
- [Use the ServiceBinder's 1.1 framework API \(see \[Version 1.0\]\(#\)\)](#).
- [Automating Service Dependency Management in a Service-Oriented Component Model](#), ICSE CBSE6 Workshop, 2003.
- [ServiceBinder tutorial](#): This tutorial explains the basics of building applications with OSGi and the benefits of using the ServiceBinder.

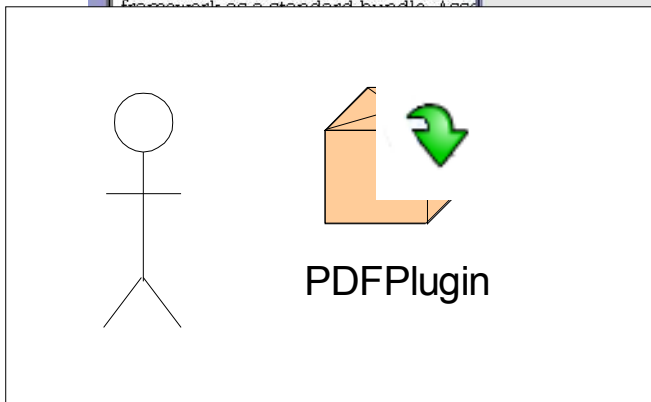


Gestionnaire de composition

The screenshot shows a web browser window with two tabs. The active tab is titled 'Web Binder' and displays the URL `http://www-adele.imag.fr/~cervante/servicebinder/`. The page content includes the title 'ServiceBinder', the subtitle 'Simplifying application development', and authors 'Humberto Cervantes, Richard...'. A 'Summary' section is visible, starting with 'The ServiceBinder is a mechanism that automates service dependency management...'. One of the most complex tasks faced by this platform is writing assembly and... oriented environment. Inside such an... (i.e., they arrive or depart at any time... capable of handling these situations. A... responsible for writing both application... general, complex, as it requires monitor... these two types of logic end up interm... difficult. The ServiceBinder solves this problem... the bundles and moving it into an exec... framework as a standard bundle. App...

Overlaid on the browser is an 'Error Occurred' dialog box. The message reads: 'An error has occurred during the loading of the requested page.' Below this, the 'Error Details' section shows a stack trace: `com.javio.webwindow.N at com.javio.webwindow.ZI.addTabable(Unknown Source) at com.javio.webwindow.ZI.build(Unknown Source) at com.javio.webwindow.ZI.build(Unknown Source) at com.javio.webwindow.OI.load(Unknown Source) at com.javio.webwindow.OI.run(Unknown Source) at F.C.run(Unknown Source)`

At the bottom of the browser window, the URL `http://www-adele.imag.fr/BEANOME/papers/CervantesHallCBSE2003.pdf` is visible.





Gestionnaire de composition

The screenshot shows two overlapping web browser windows. The background window displays the 'ServiceBinder' website, which includes a title, authors (Humberto Cervantes, Richard S. Hall), and a summary section. The foreground window displays a PDF document titled 'Automating Service Dependency Management in a Service-Oriented Component Model' by Humberto Cervantes and Richard S. Hall. The PDF content includes an abstract, keywords, and the start of an introduction.

ServiceBinder
Simplifying application development
Authors: [Humberto Cervantes](#), [Richard S. Hall](#)

Summary

The ServiceBinder is a mechanism that automates service dependency management in a service-oriented component model.

One of the most complex tasks faced by developers on this platform is writing assembly and deployment code in a service-oriented environment. Inside such an environment, services (i.e., they arrive or depart at any time) and components (i.e., they are capable of handling these situations). Applications are responsible for writing both application logic and deployment logic. In general, complex, as it requires monitoring and managing these two types of logic end up intertwined and difficult.

The ServiceBinder solves this problem by automating the bundles and moving it into an execution framework as a standard bundle. Assesment of the information contained in an [XML description](#) of the application. Applications built with the ServiceBinder can adapt themselves autonomously, for example, by integrating new services that arrive dynamically.

The ServiceBinder is small (~70k) and easy to use.
[Link](#)

Automating Service Dependency Management in a Service-Oriented Component Model
Humberto Cervantes and Richard S. Hall
Laboratoire LSR Imag, 220 rue de la Chimie
Domaine Universitaire, BP 53, 38041
Grenoble, Cedex 9 France
{Humberto.Cervantes,Richard.Hall}@imag.fr

ABSTRACT
This paper describes a mechanism to automate service dependency management in a service-oriented component model. The impetus behind this mechanism is not merely to eliminate complex and error-prone code from component-based applications, but also to deal with the phenomena of application building blocks that exhibit dynamic availability, i.e., they may appear or disappear at any time and this is not under the control of the application. This intense focus on dynamic availability of building blocks is the result of the belief that applications of the future will become context aware in order to deal with building block proliferation. Such applications will employ context-aware architectures that use context (e.g., location, environment, user task) as a filter for including/excluding building blocks in/from their compositions. In this vision, automatic handling of dynamically available building blocks and their impact on application composition is critical. The service dependency management mechanism described in this paper is a starting point for such research and is implemented on top of the Open Services Gateway Initiative (OSGI) framework. The concepts and solutions it provides are sufficiently general for application in other service-oriented component models.

Keywords
Service-Oriented Programming, Components, OSGi

1. INTRODUCTION

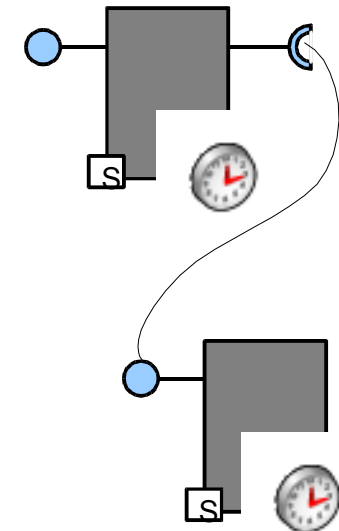
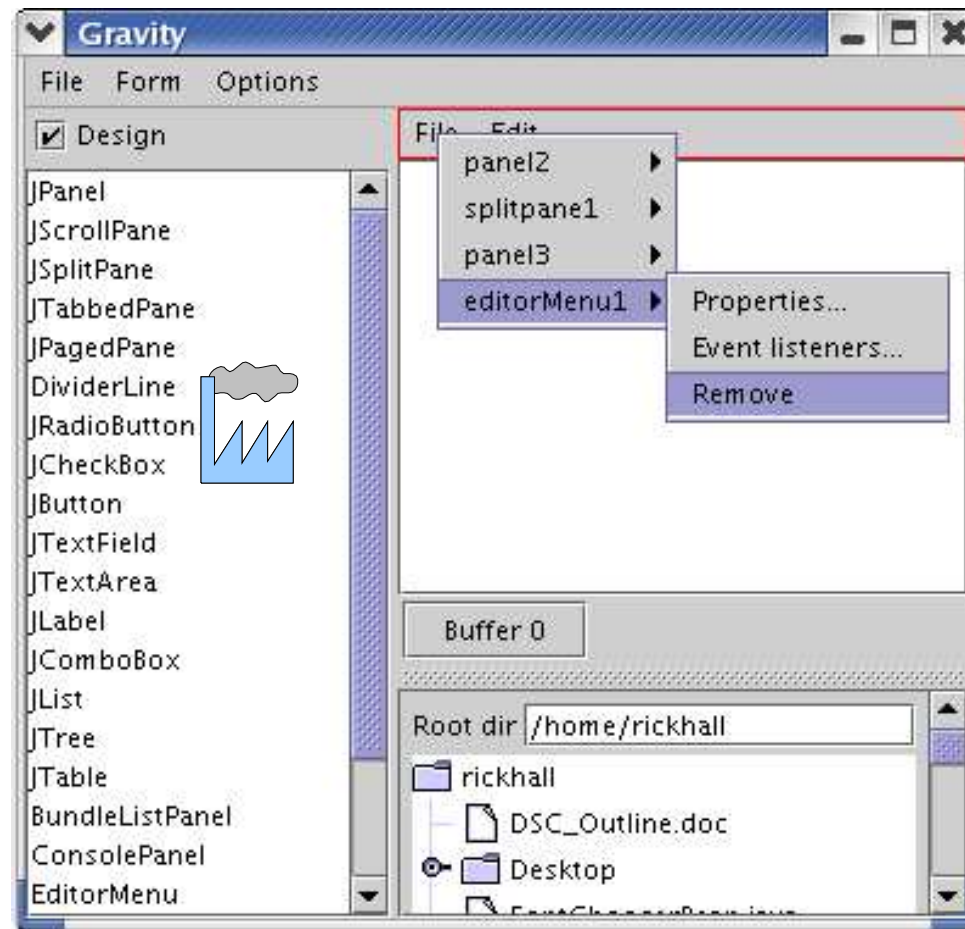
based services and as a result push the inherent unreliability of distributed systems into ordinary client-side applications. Pervasive computing strives to embed computing power into almost all imaginable devices, each of which is able to offer services via wireless networks and other protocols. In both of these cases, service failures may occur, for example, when a server crashes or when a user simply walks out of wireless network range. These types of occurrences require that applications using the failed services deal with their dynamic departure. Likewise, applications may have to deal with dynamic building block arrival when servers or network connections are restored or when completely new services are discovered.

These scenarios are relevant to modern-day computing systems, but they also foreshadow a future in which continuous network connectivity is common and building blocks proliferate beyond the ability of applications to integrate efficiently and meaningfully with them. To deal with this coming building block proliferation, we envision a future where applications leverage context awareness in the form of context-aware architectures, where context (e.g., location, environment, user task) is used as a filter to determine which building blocks are included/excluded in/from an application's architectural composition at any given time. In this scenario, dynamic building block availability is the underlying issue to be resolved. Building blocks appear/disappear to/from an application based on their relevance to the current context. In turn, the application's composition must automatically adapt to these changes. Changes in context and the building blocks

Page 1 / 6 - PDFGo.com DEMO VERSION - See <http://www.pdfgo.com> for licensing.
[Link: Delegating to plugin...](#)



Gravity



Assemblage



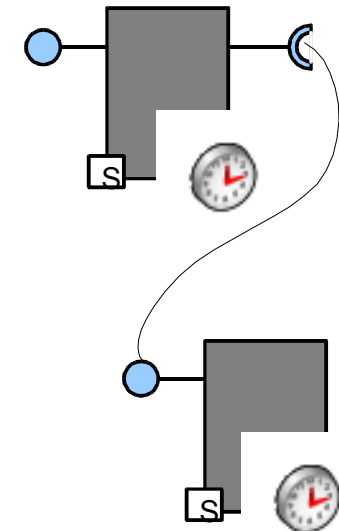
Gravity

```
Gravity
File Edit
import java.io.InputStream;
import java.net.URL;
import java.util.Dictionary;

public interface Bundle
{
    public static final int UNINSTALLED = 1;
    public static final int INSTALLED = 2;
    public static final int RESOLVED = 4;
    public static final int STARTING = 8;
    public static final int STOPPING = 16;
    public static final int ACTIVE = 32;
}

Buffer 0 Buffer 1

Root dir /home/rickhall/projects/win-projects/oscar/src/org
org
├── mortbay
├── osgi
├── framework
└── AdminPermission.java
```



Exécution

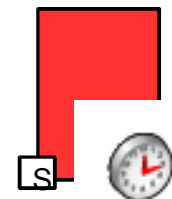
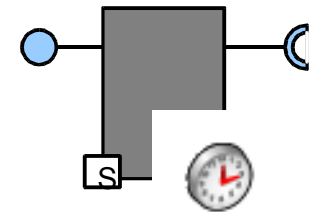



Gravity

```
Gravity
File Edit
import java.io.InputStream;
import java.net.URL;
import java.util.Dictionary;

public interface Bundle
{
    public static final int UNINSTALLED = 1;
    public static final int INSTALLED = 2;
    public static final int RESOLVED = 4;
    public static final int STARTING = 8;
    public static final int STOPPING = 16;
    public static final int ACTIVE = 32;
}
```

Buffer 0 Buffer 1



Exécution en mode dégradé



Plan de la thèse

Problématique

Fondations

Modèle à composants orienté services

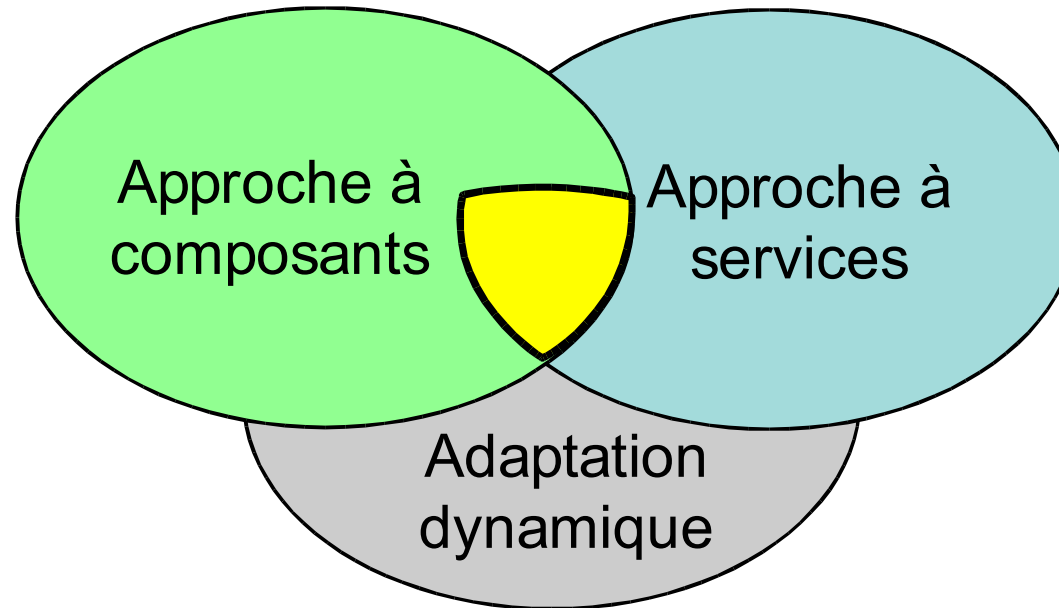
Réalisation

Évaluations

Conclusion et perspectives



Synthèse



Modèle à composants orienté services

- Construire des applications supportant la disponibilité dynamique
- Support d'exécution de ces applications



Synthèse

Concepts du modèle à composants orienté services

- Composants à services
- Instances valides et invalides
- Gestionnaires d'instances
- Instances de déploiement et dynamiques
- Espaces de résolution
- Descripteur de composition

Applications capables

- D'auto-assemblage
- D'auto-adaptation à travers l'incorporation, retrait et substitution de fonctionnalités
- De supporter le déploiement continu



Conclusions

Approche se révèle très adéquate

- « *Le tout plus grand que la somme des parties* »
- Nouveaux défis comme l'imprévisibilité

Approche indépendante de modèle à composants et de plate-forme de services

- Autres implémentations possibles

Idées réutilisables dans la programmation orientée services en général

- Services != Services Web



Conclusions

Réalisation

- OSGi est une base adéquate pour construire un modèle à composants
- Rajout de concepts à la plate-forme OSGi, tout en restant compatibles
- Regles d'adaptation simples facilitent adoption
- Retours sur l'utilisation du Service Binder très encourageants

Environnement d'exécution extensible

- Construction d'un noyau orienté services?
- Déploiement continu

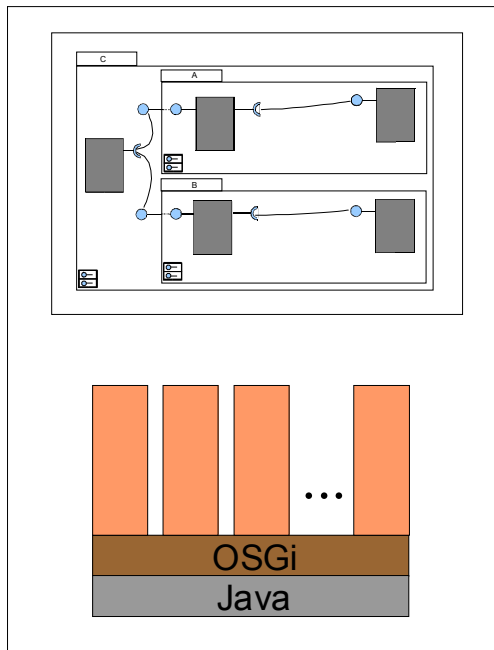


Perspectives

Mécanismes de description et de découverte de services plus complexes

- Ajout d'informations sémantiques

Construction d'applications sensibles au contexte



- Emplacement utilisateur
- Interactions utilisateur
- Disponibilité réseau
- etc...

Questions ?



TEMPLATE
