



LSR-Adèle  
Université Grenoble I

# Autonomous Adaptation to Dynamic Availability Using a Service-Oriented Component Model

Humberto Cervantes  
Richard S. Hall

---

ICSE, May 28th 2004



# Motivation

---

## **Introduce and support dynamic availability in component-based applications**

### **What is dynamic availability ?**

- Situation where functionality provided by components that is being used or that could be used by an application becomes available or unavailable during execution, potentially outside of its control

### **A trend that is becoming more common**

- Extensible systems, continuous deployment, wireless connectivity, web services, context-driven applications

### **However**

- Not explicitly supported in existing component frameworks



# Component Orientation

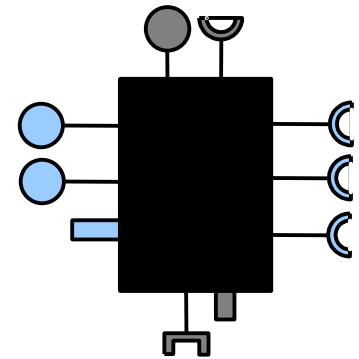
---

## Founding principles

- Applications are built by composing (i.e., assembling) reusable building blocks called *components*
- Component development and application assembly are activities that are clearly differentiated

## Component characteristics (Szyperski)

- Binary unit of composition
- Contractually defined interfaces
- Explicit dependencies
- Independent delivery and deployment

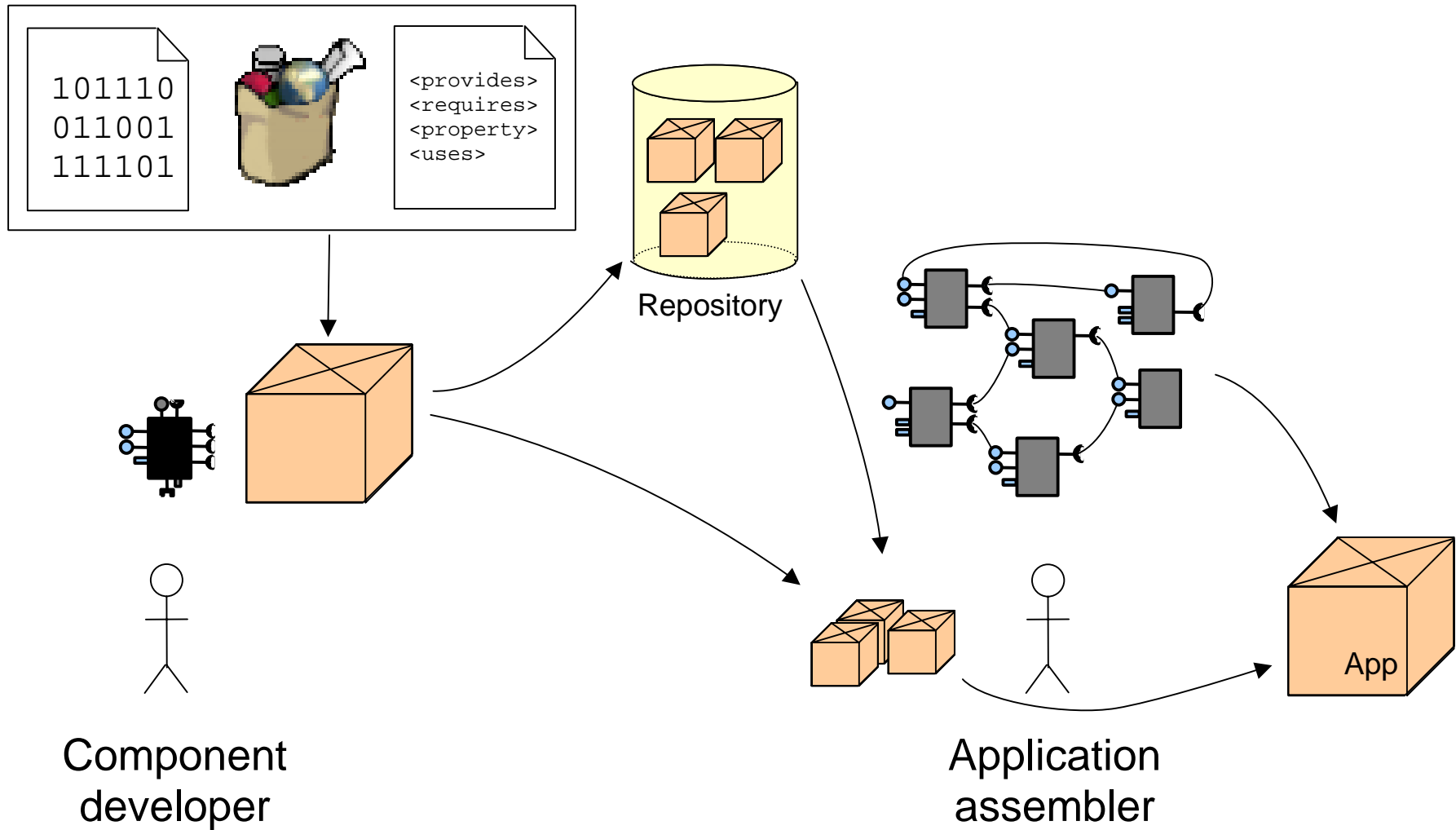


## But also...

- Separation between package, component, and instance



# Component-Based Development





# Service Orientation

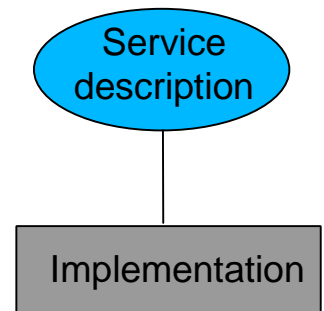
---

## Founding principles

- Applications built by composing reusable building blocks called *services*
- Service development and application assembly are activities that are clearly differentiated

## Services

- Functionality that is contractually defined
- Description : syntactic + semantic information

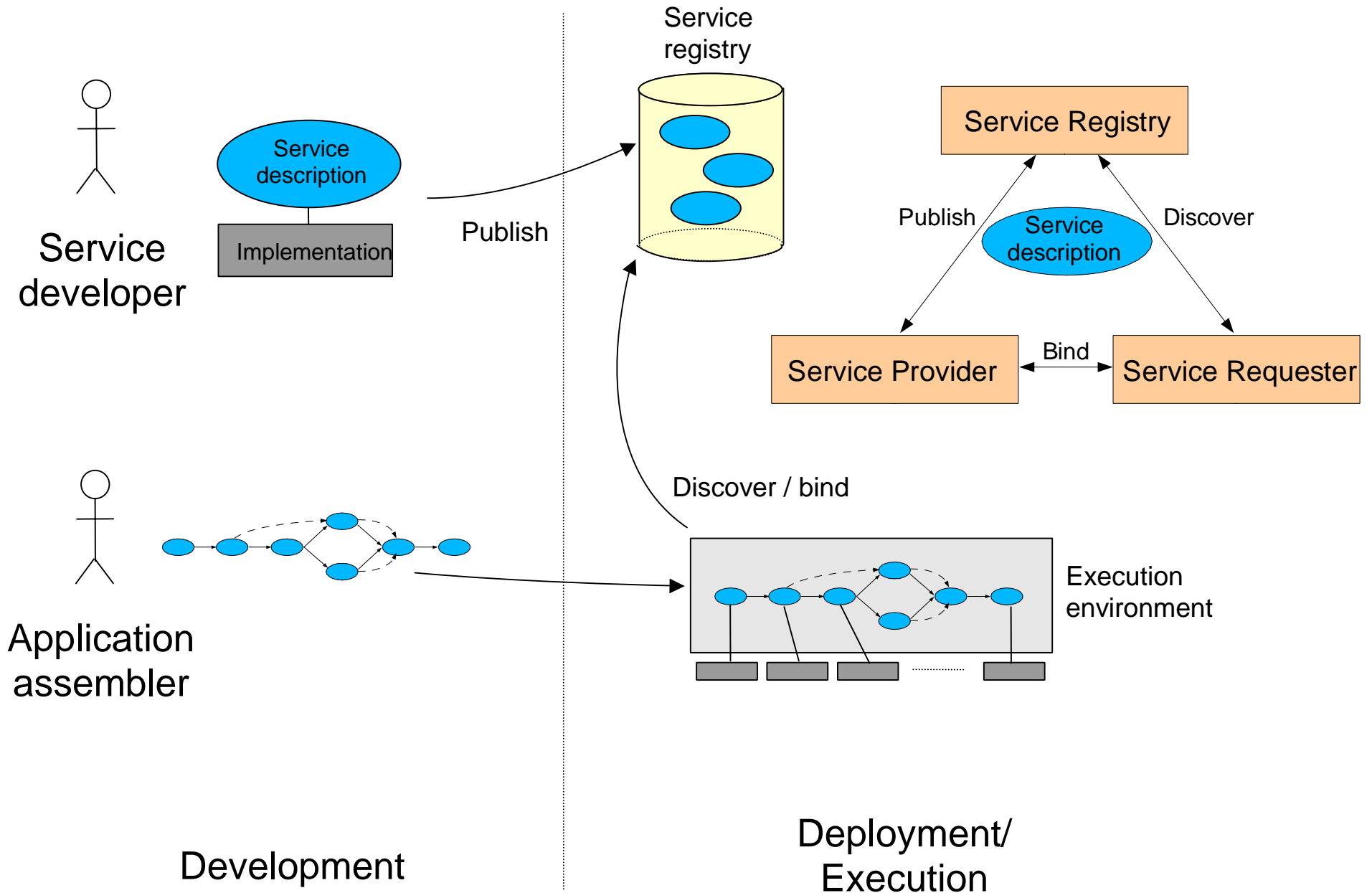


## Late integration

- Late discovery of service providers, prior to or during application execution
- Use of the service-oriented interaction pattern (trading)
- Substitutability and dynamic availability are hypothesis



# Service-Oriented Development





# Differences Between Approaches

---

## **Building block integration time**

- At the time of assembly in component orientation
- Prior to or during execution in service orientation

## **Dynamic availability**

- Not an explicit assumption in component orientation
- Present in service orientation

## **Other differences**

- Packaging and deployment not really considered in service orientation
- Concept of instantiation not present in service orientation



# The Gravity Framework

---

**A framework to build and execute component based applications where dynamic availability is present**

- Dynamic availability occurs as a result of continuous deployment activities

**Service-oriented component model**

- Introduces concepts from service orientation into a component model

**Execution environment**

- Provides adaptation logic that enables applications to exhibit autonomous adaptation properties
- Targets non-distributed client-side applications





# Service-Oriented Component Model

---

**Introduces concepts from service orientation into a component model**

## **Principles**

- Component description is used as a contract, similar to a service description
- Component instances provide and require services
- The service-oriented interaction pattern is used as a means to bind component instances and to manage compositions at run time

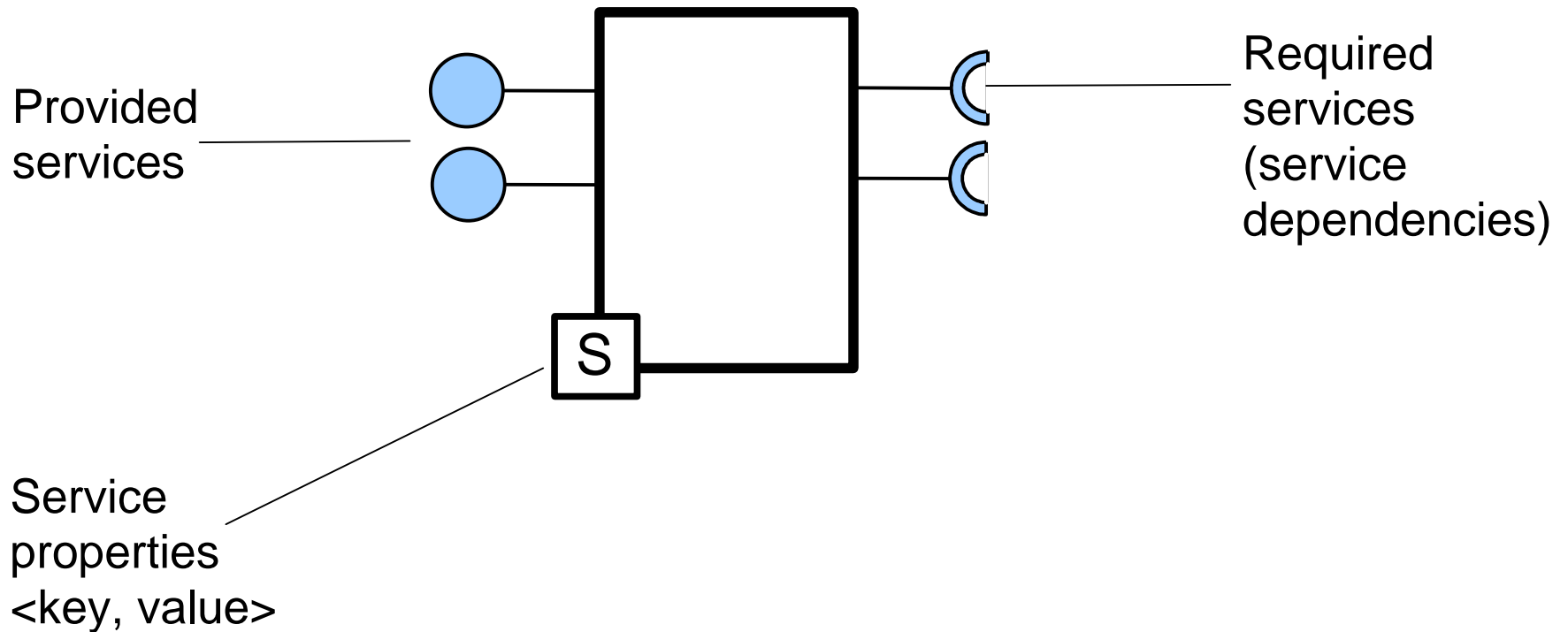
## **Challenges**

- Adaptation to dynamic changes
- Dealing with ambiguity



# Service Component

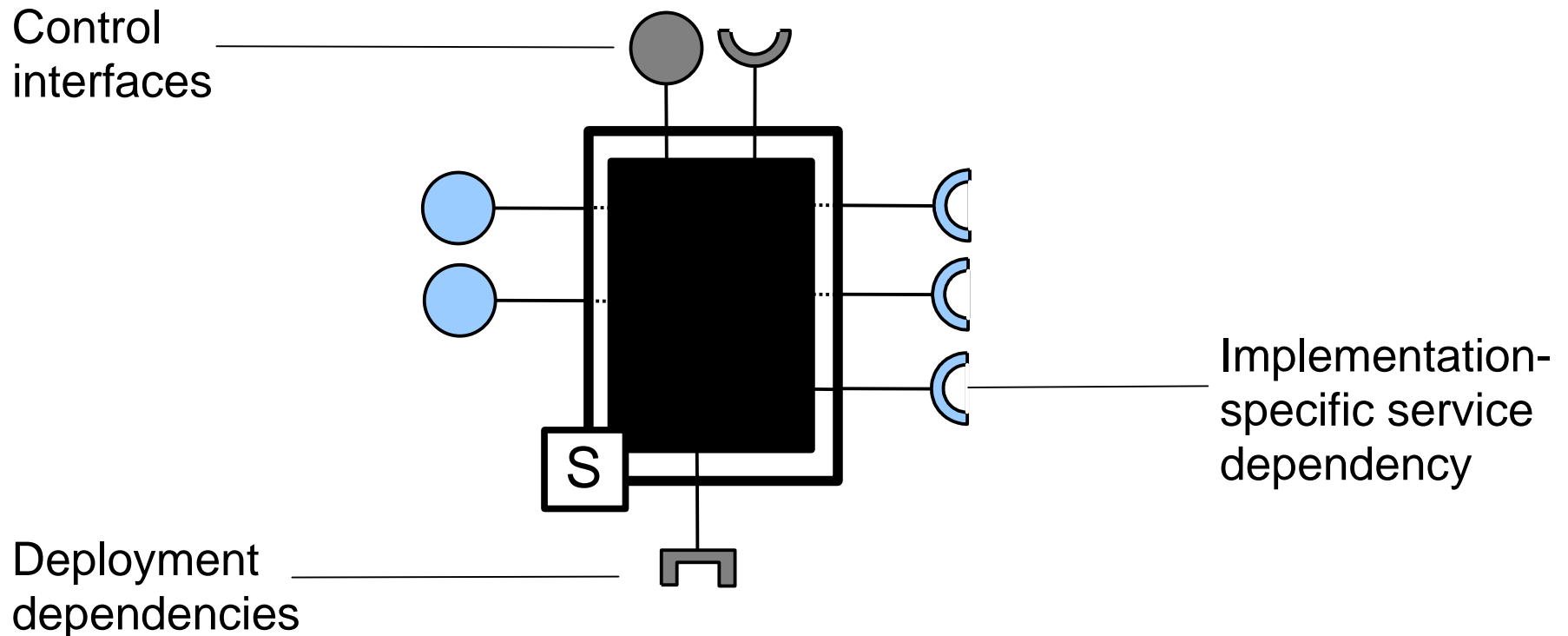
## Component description used as a contract





# Service Component

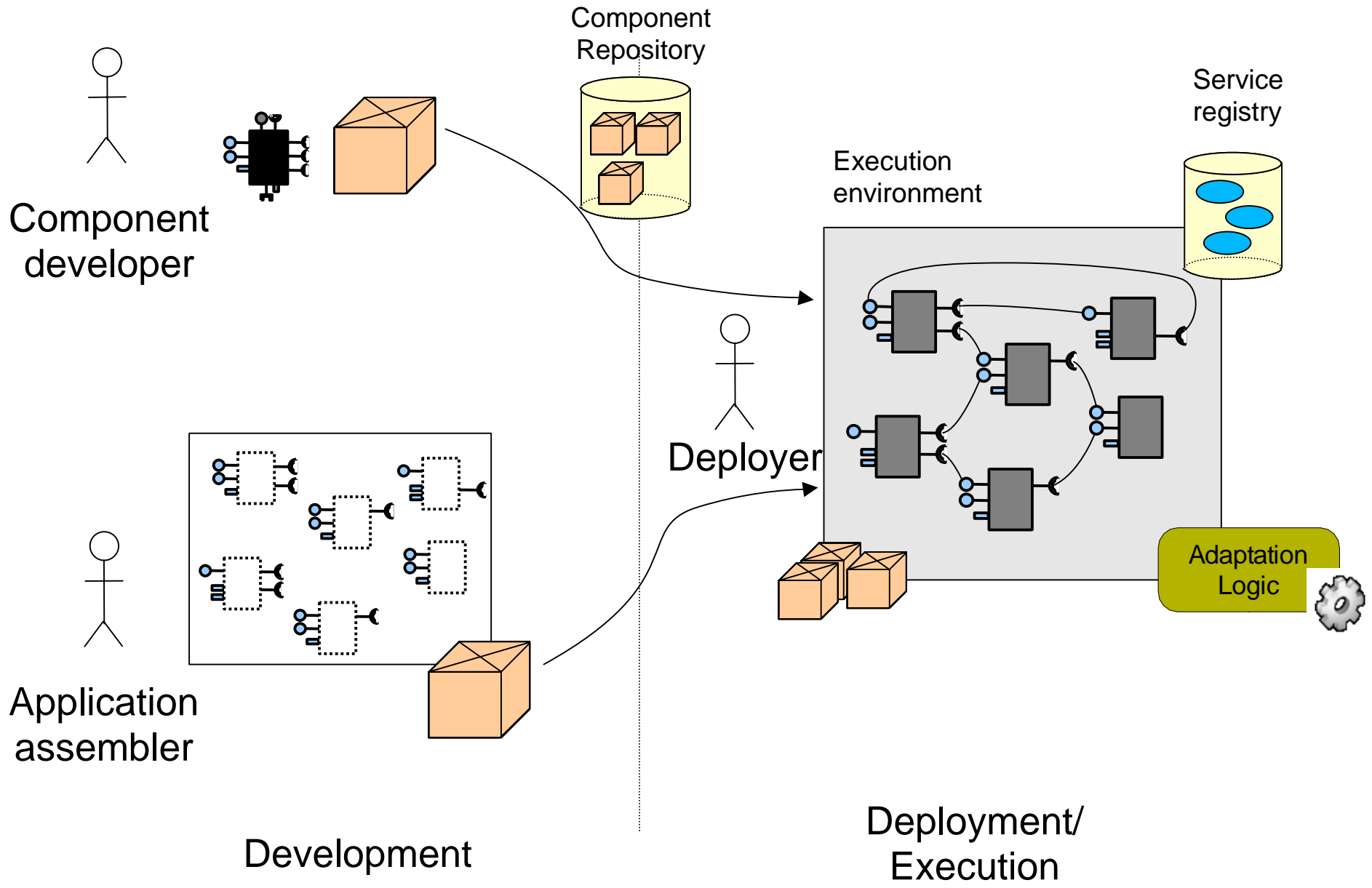
## Contract fulfilled by one or more implementations



- An implementation can provide additional services or have additional service dependencies



# Gravity's SOCM Development





# Adaptation Logic

---

## **Execution environment manages two levels of adaptation logic:**

### **Instance level**

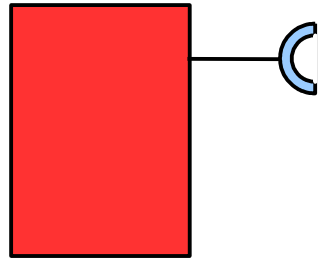
- Management of a single instance with respect to its service dependencies
- Supports the arrival and departure of component instances and their provided services

### **Composition level**

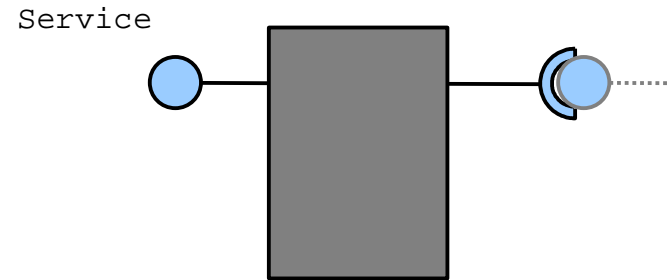
- Management of a composition with respect to a set of instances inside a scope
- Supports the arrival and departure of components



# Instance-Level Management (1)



***invalid*** instance



***valid*** instance

## Instances can be invalid or valid

- Invalid = dependencies not satisfied, services not provided
- Valid = dependencies satisfied, services are provided

## Intentional creation

- The intention is to create a valid instance, but validity is not guaranteed when instance is created
- The framework will try to honour the intention based on service dependency properties



# Instance-Level Management (2)

## Service Dependency Properties

### Name

- Java interface

### Cardinality

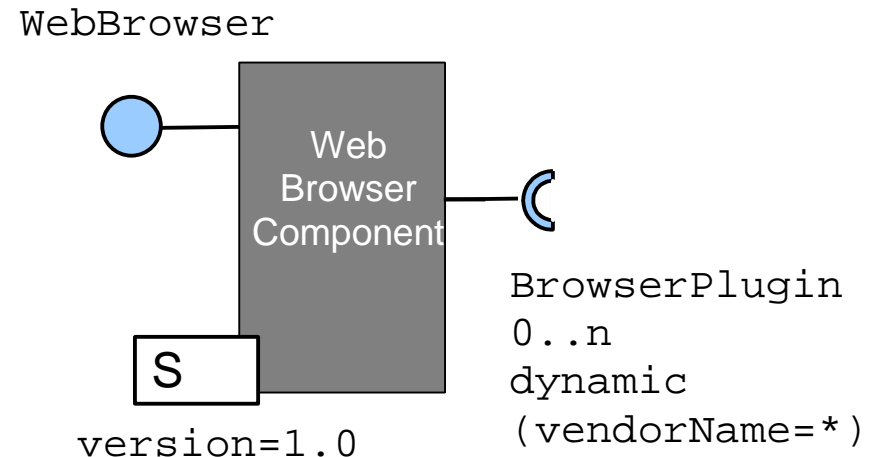
- Optional, mandatory
  - $0..*$ ,  $1..*$
- Singular, aggregate
  - $*..1$ ,  $*..n$

### Policy

- Static, dynamic

### Filter

- Boolean expression consisting of service property values

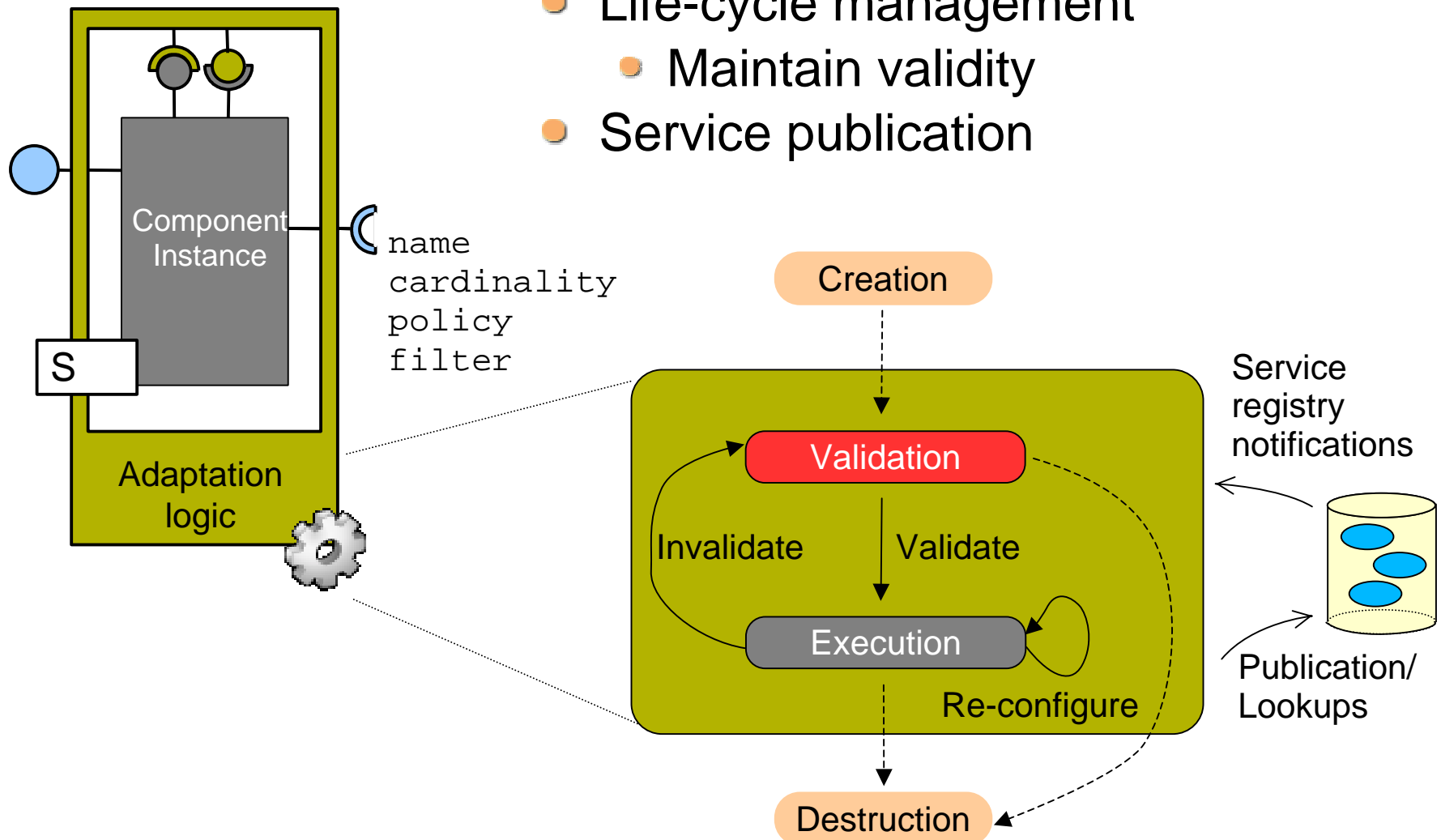




# Instance-Level Management (3)

The instance manager is responsible for

- Dependency management
- Life-cycle management
  - Maintain validity
- Service publication







# Instance-Level Management (4)

---

## **Can be used to create certain types of applications**

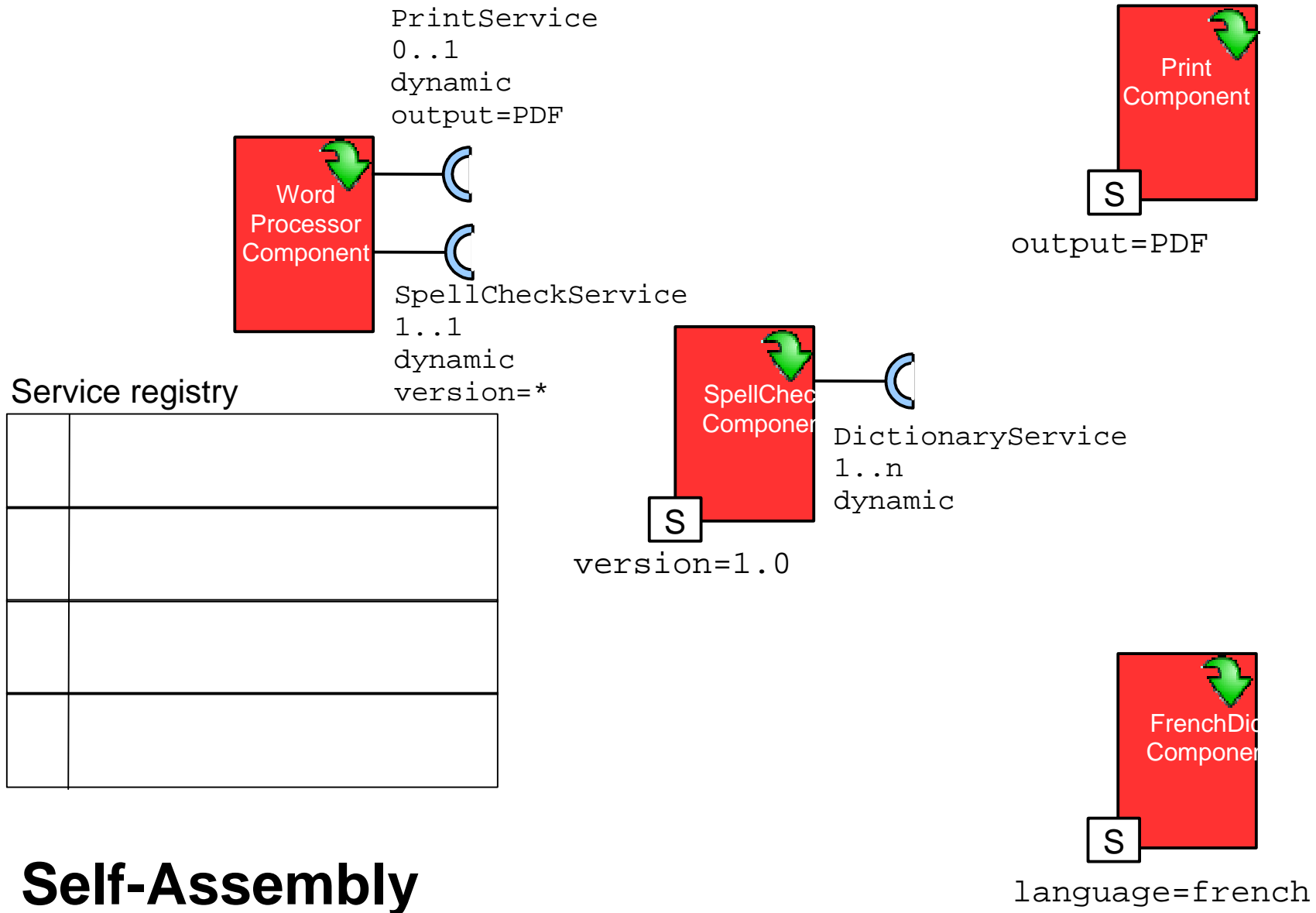
- Typically built out of singletons that are created as a result of deployment activities
- Core component instance uses or provides services.

## **These applications are capable of automatic**

- Self-assembly
- Functionality integration
- Functionality removal
- Functionality substitution



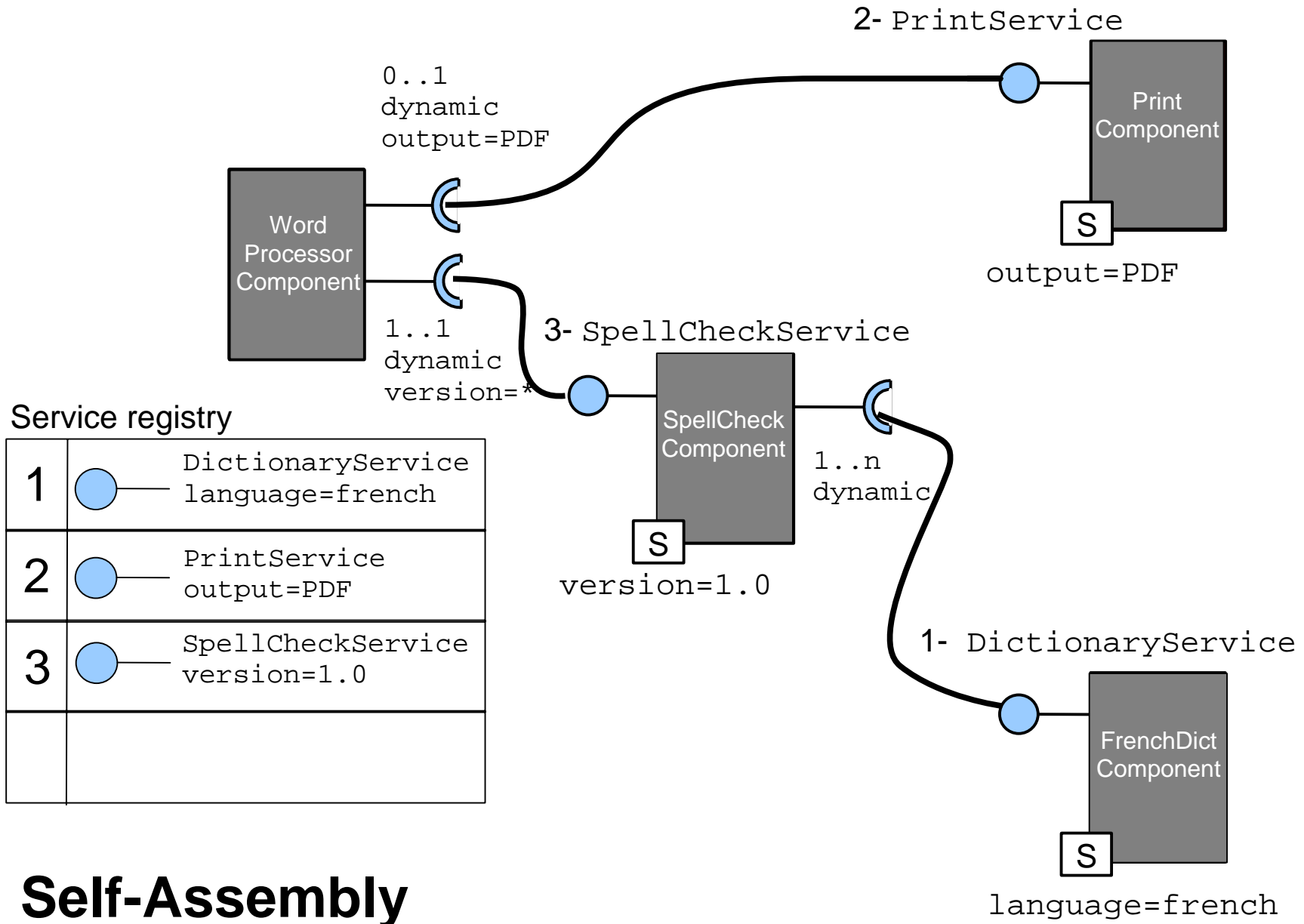
# Instance-Level Management (5)



**Self-Assembly**



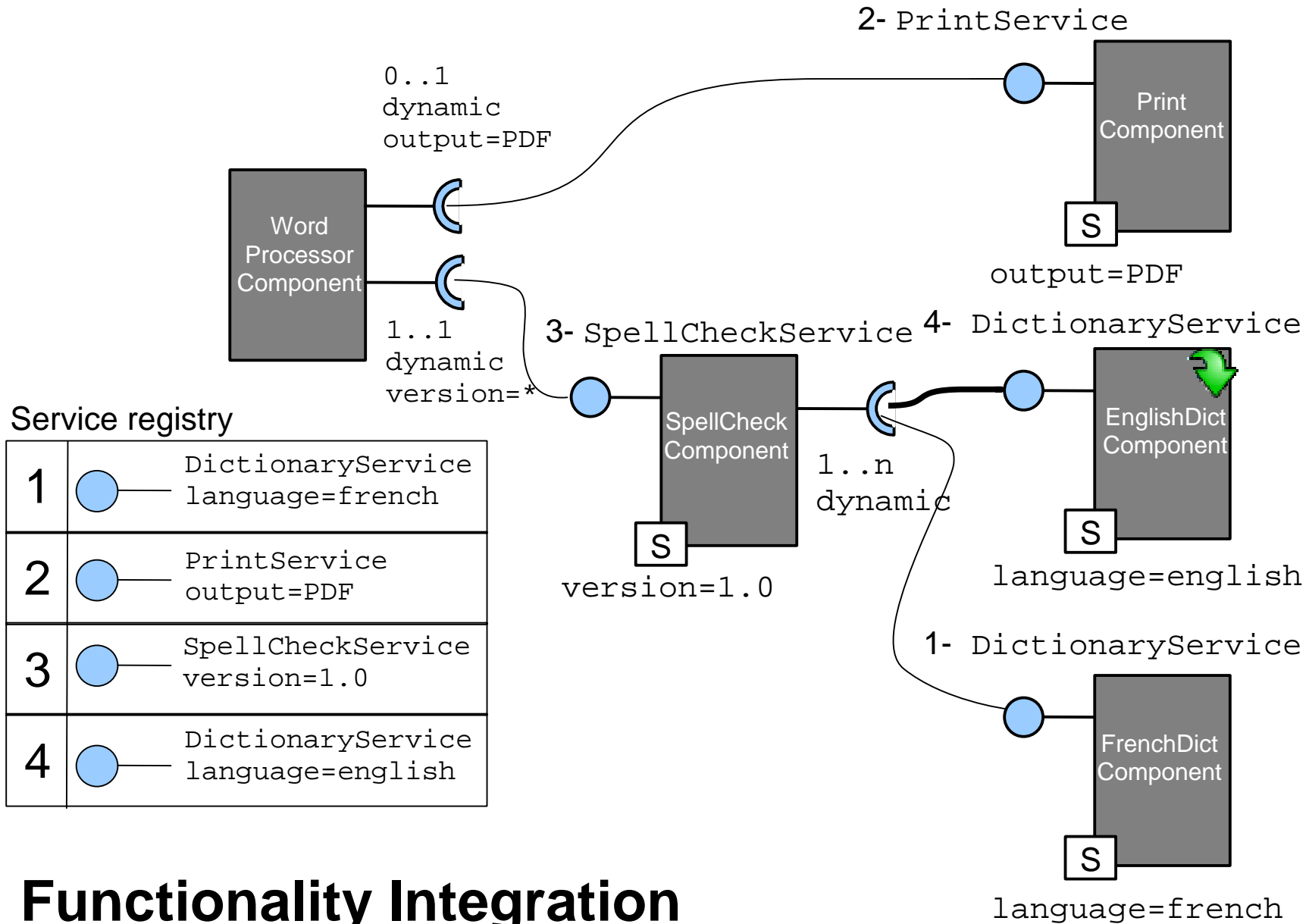
# Instance-Level Management (5)



**Self-Assembly**



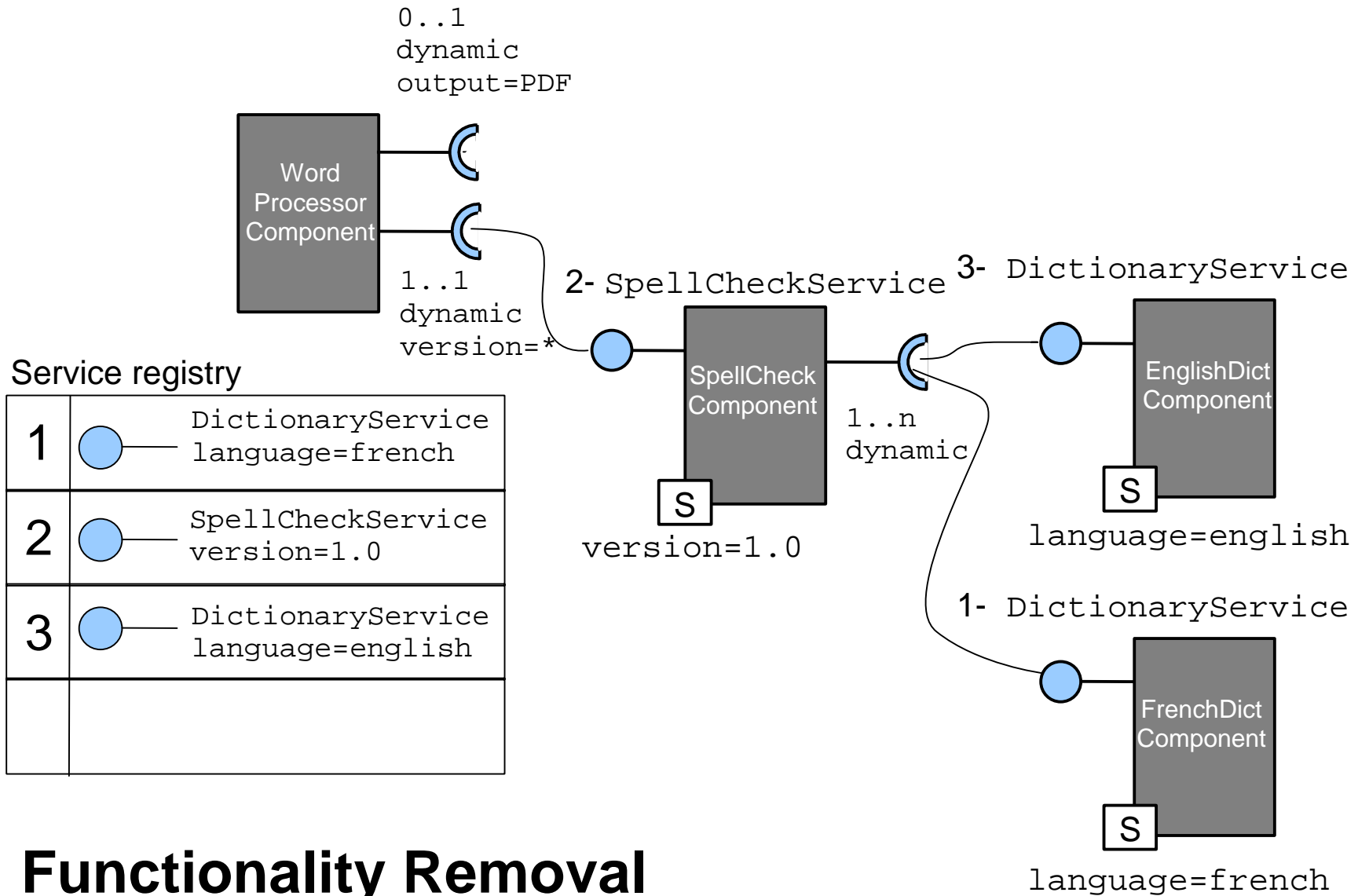
# Instance-Level Management (5)



## Functionality Integration

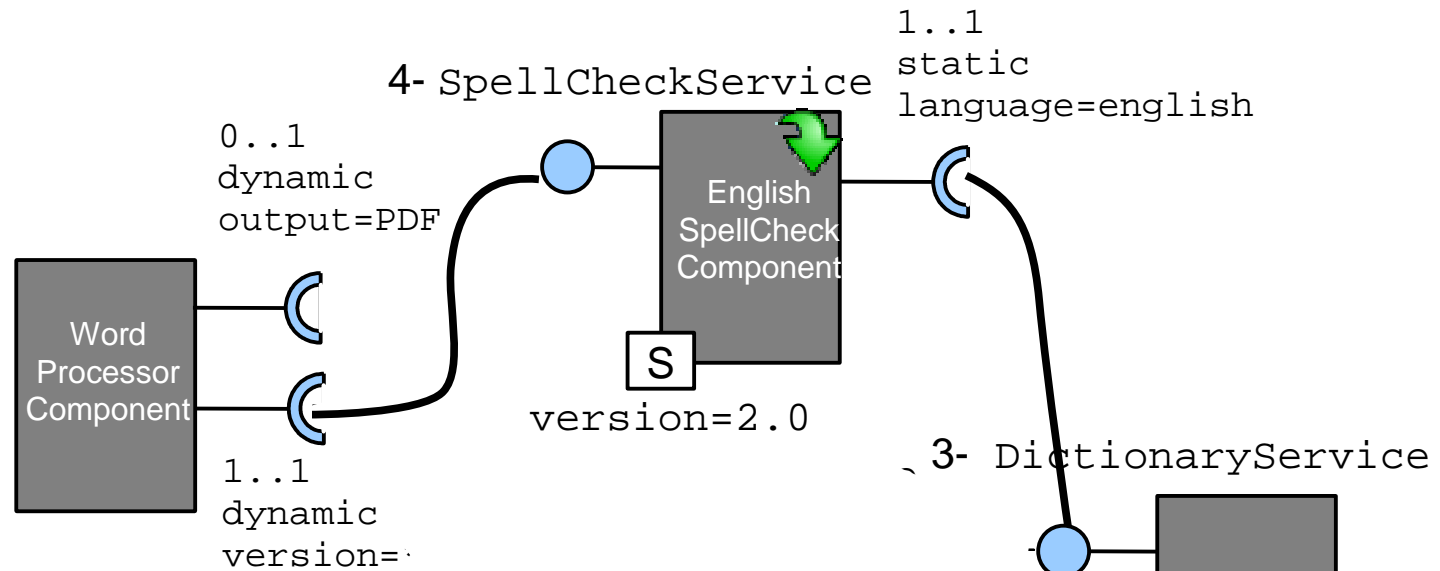


# Instance-Level Management (5)





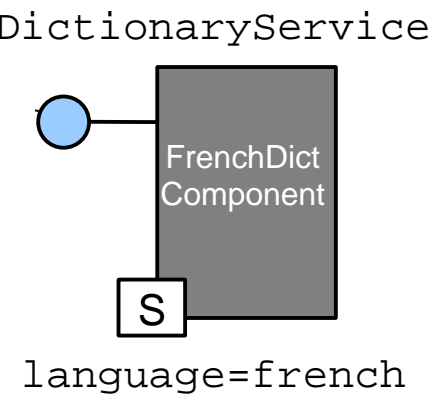
# Instance-Level Management (5)



Service registry

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

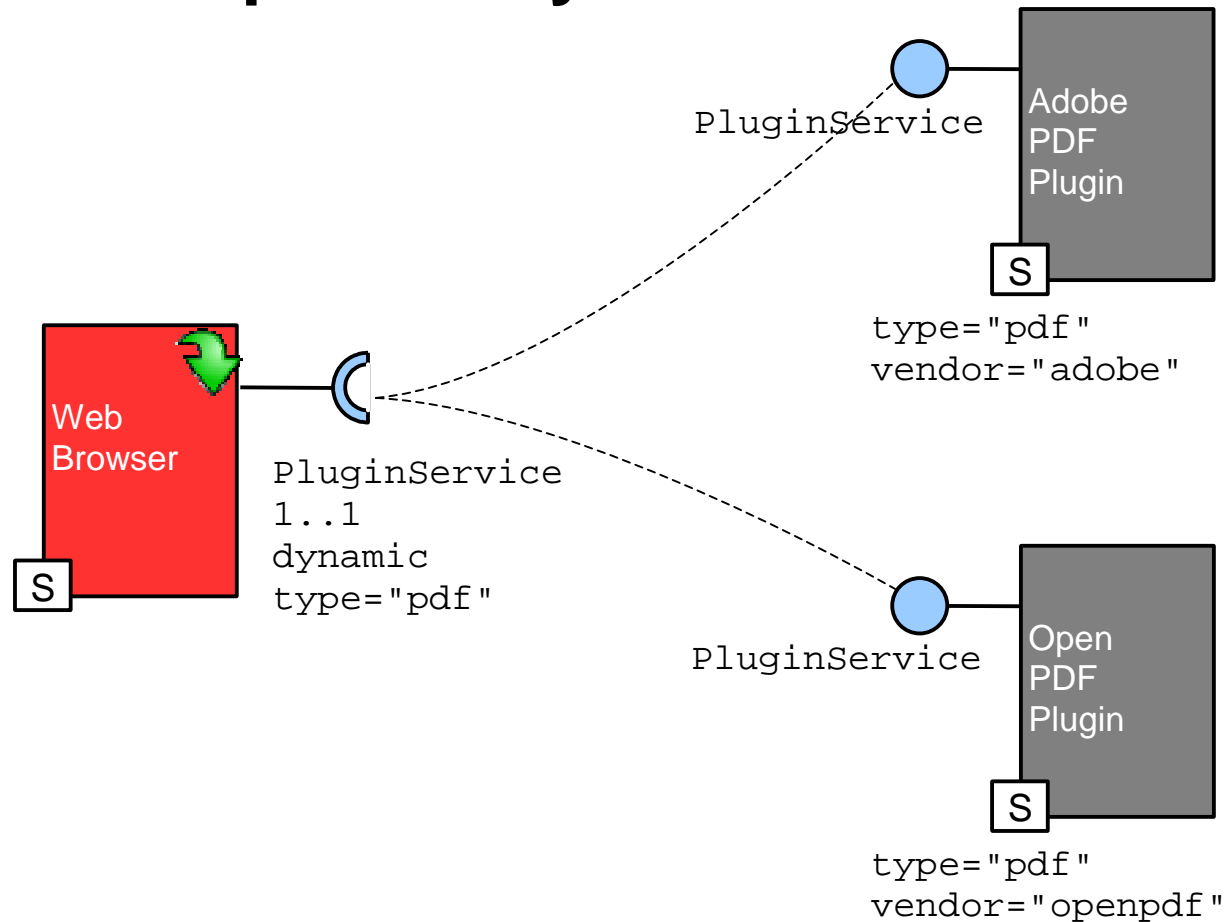
## Functionality Substitution





# Instance-Level Management (6)

**Ambiguity occurs when multiple candidates fulfil a service dependency**

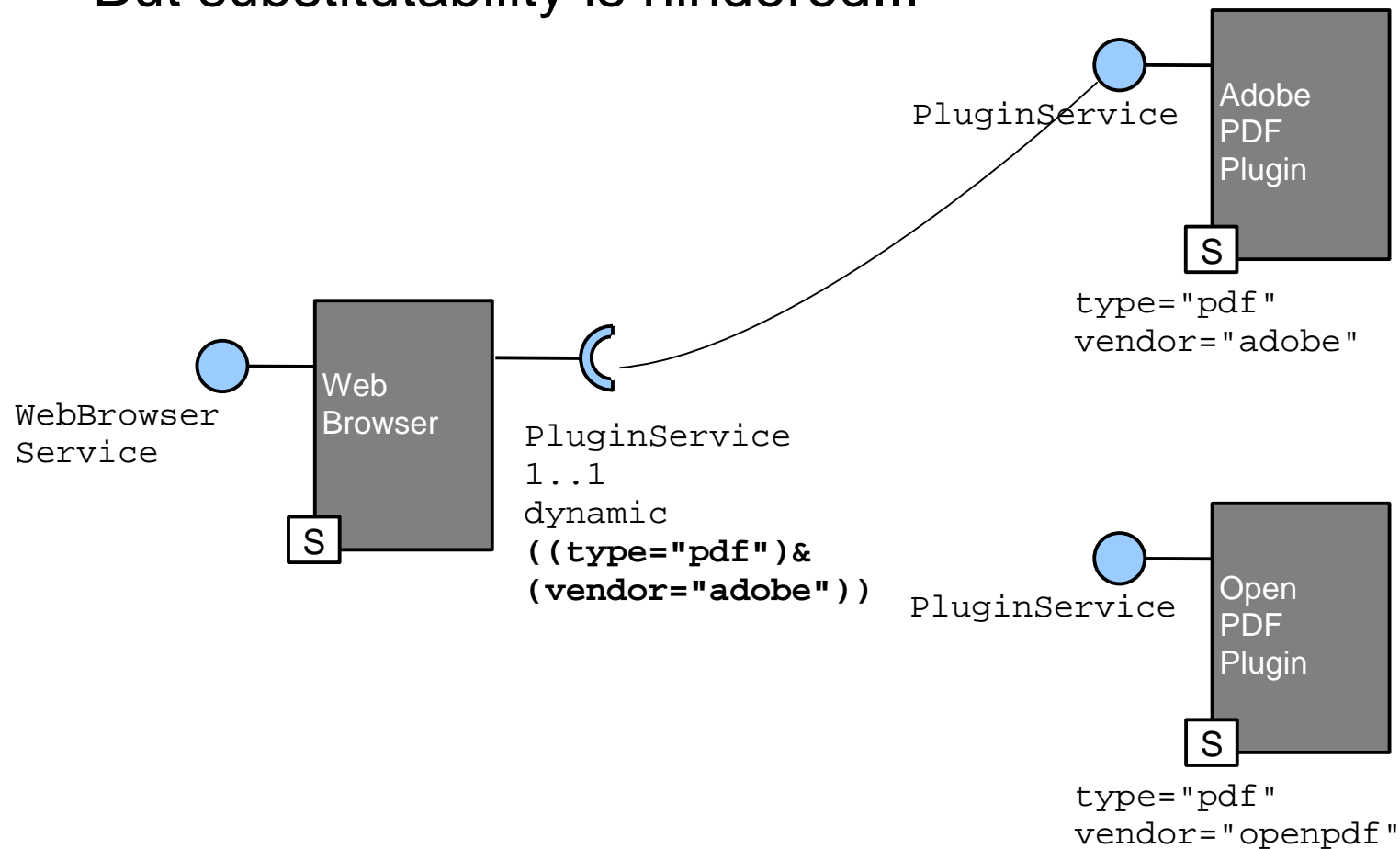




# Instance-Level Management (6)

## Ambiguity can be limited by using filters

- But substitutability is hindered...

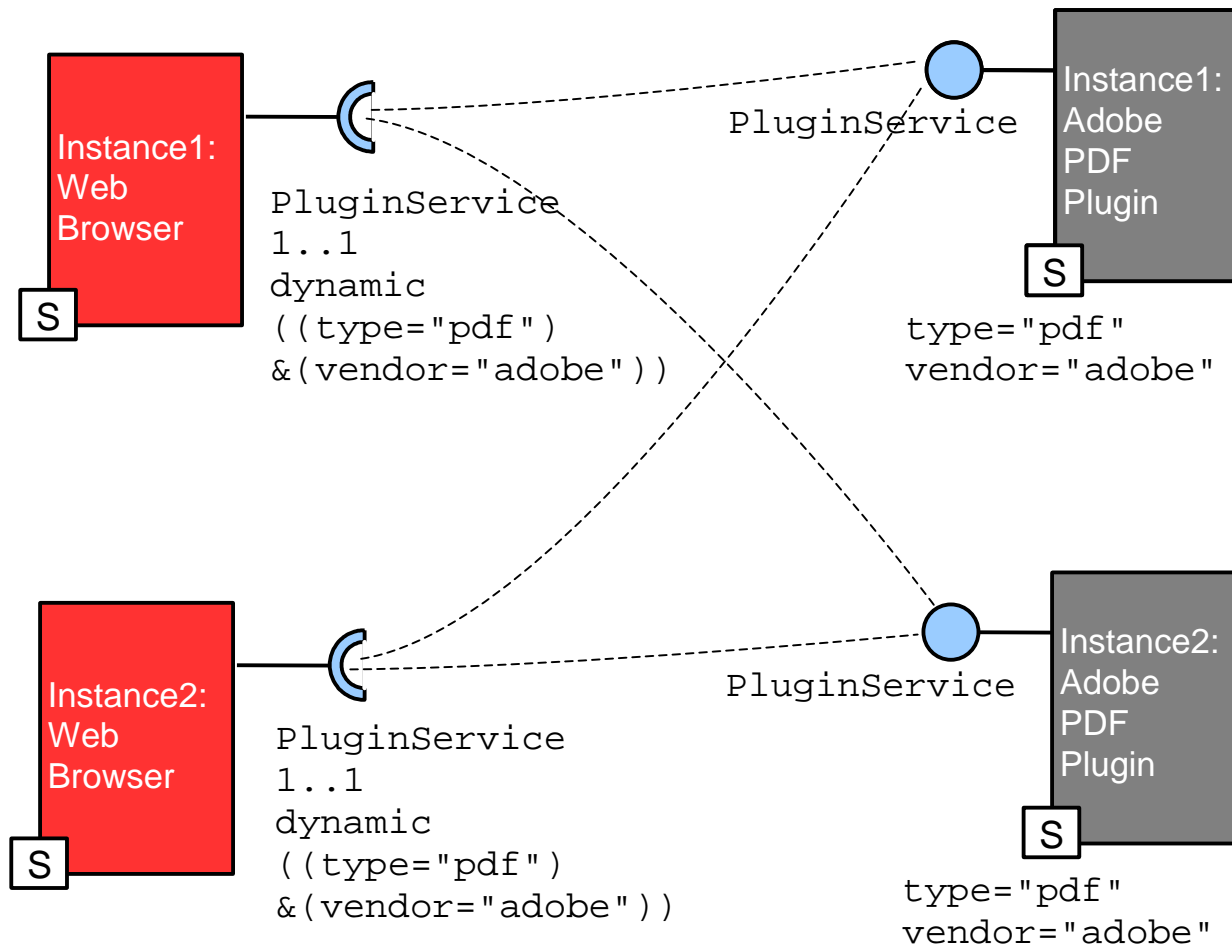






# Instance-Level Management (6)

## Filters are not useful when multiple instances exist





# Composition-Level Management (1)

---

## **Composition**

- A set of component instances created inside a scope

## **Scope**

- A well defined boundary that constrains service dependency management
- Each scope has an associated service registry
- Scopes only allow ambiguity to be limited, but scopes can also be nested

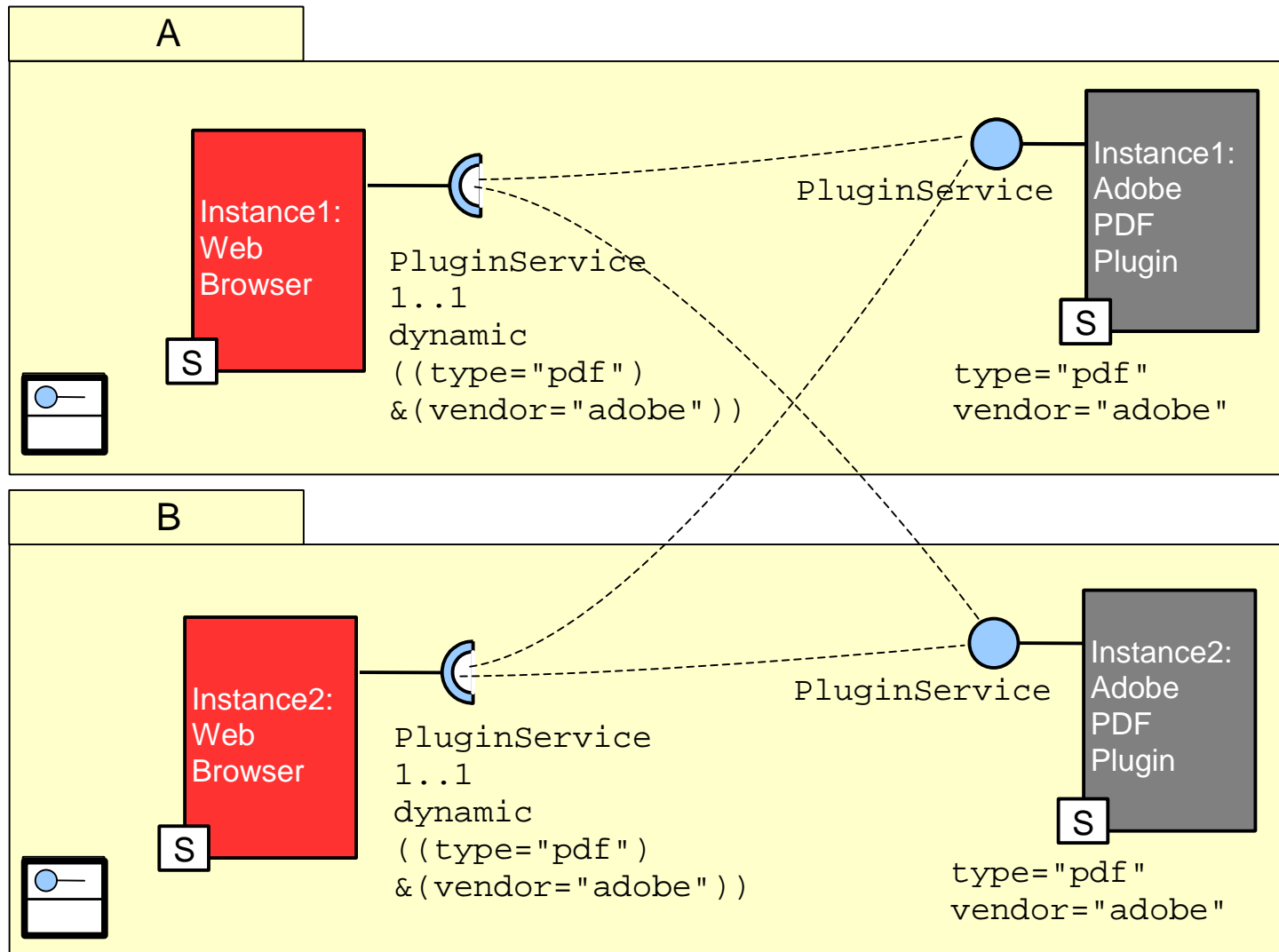
## **Composition descriptor**

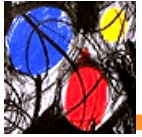
- Guides instance run-time creation of instance inside a scope



# Composition-Level Management (2)

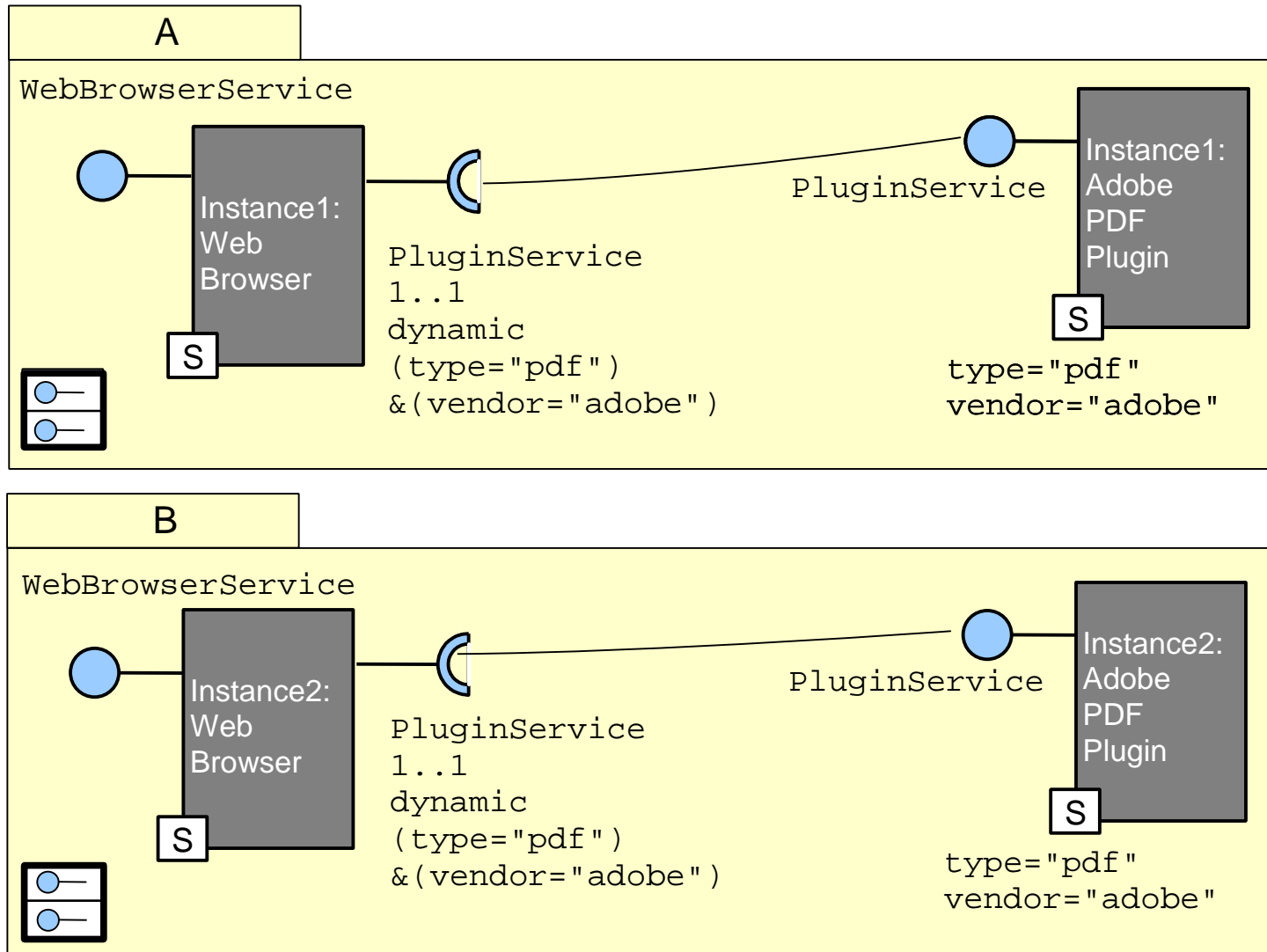
## Using scopes to limit ambiguity





# Composition-Level Management (3)

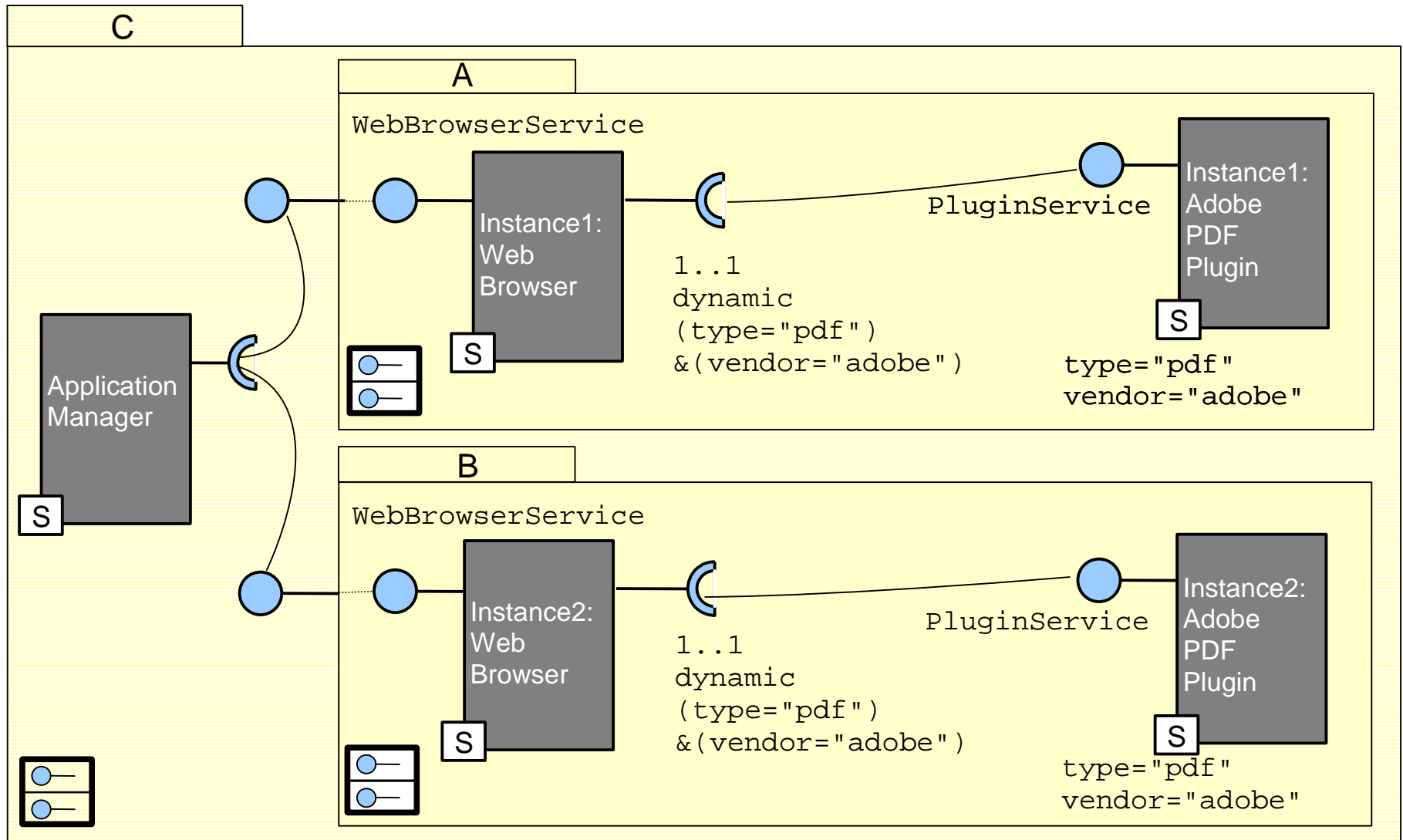
## Using scopes to limit ambiguity





# Composition-Level Management (3)

## Nested scopes result in hierarchical compositions





# Composition-Level Management (4)

---

## **A composition descriptor defines a set of instances to be created inside a scope at run time**

- Defined in terms of placeholders which allow substitutability and dynamism to be supported
- Non-explicit bindings as instance management occurs once an instance is created inside a scope

## **A placeholder represents a set of instances**

- A component contract
- Cardinality: optional/mandatory singular/multiple instances
- Filter

## **Run-time management**

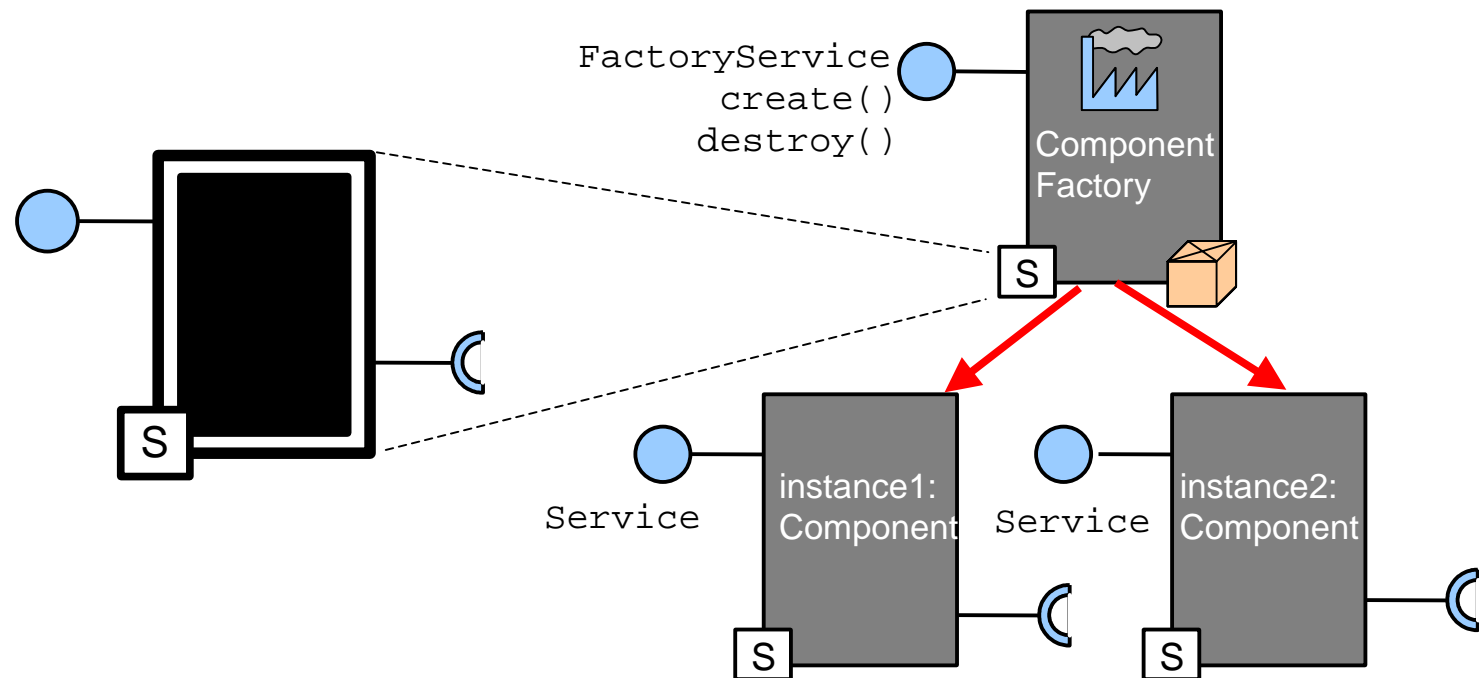
- Valid = mandatory instances can be created
- Invalid = mandatory instances cannot be created, the whole scope is destroyed
- SO interaction pattern used to locate factories



# Composition-Level Management (5)

## Component instances created through factories

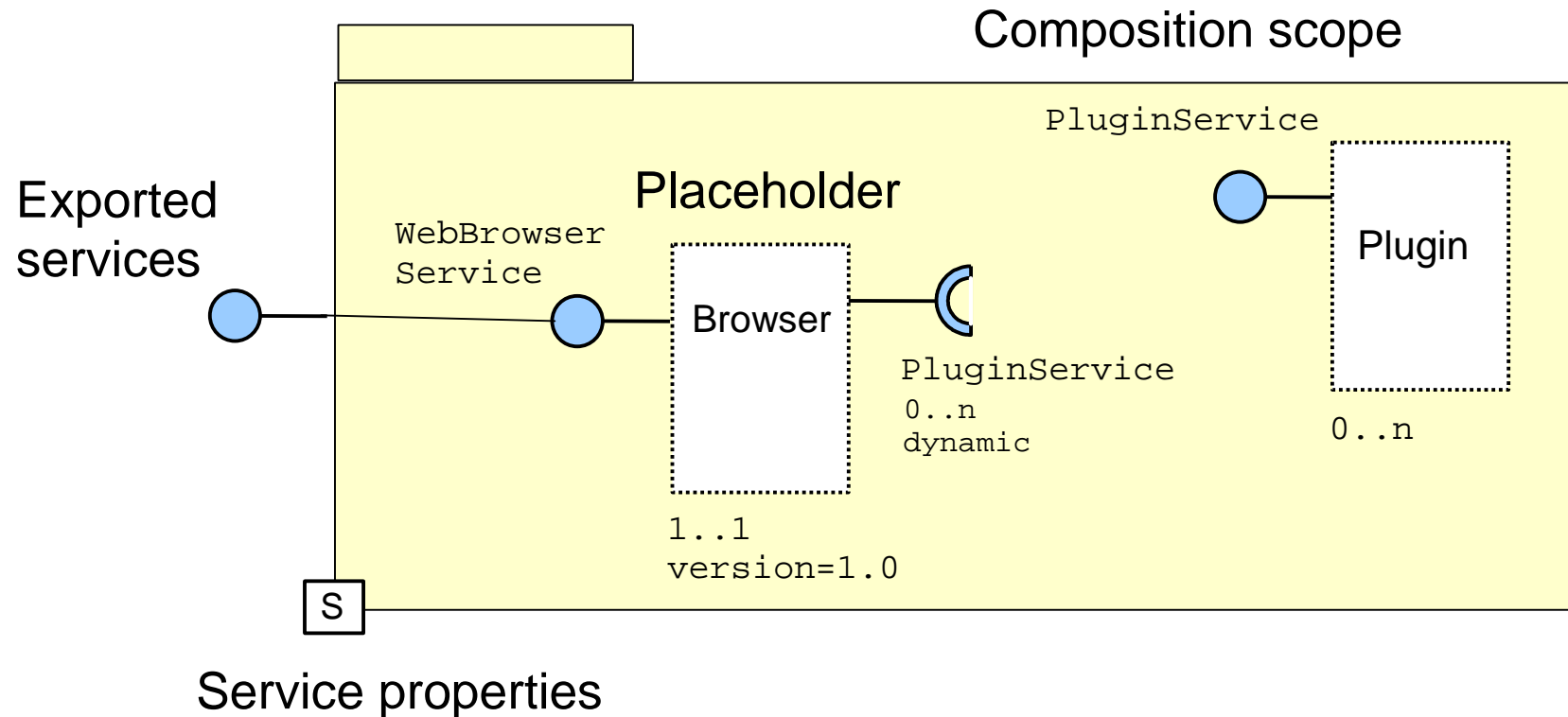
- Modeled as services provided by singleton instances
- Factories service properties contain information about the properties and the services provided and required by the component implementation





# Composition-Level Management (6)

## Elements of a composition descriptor



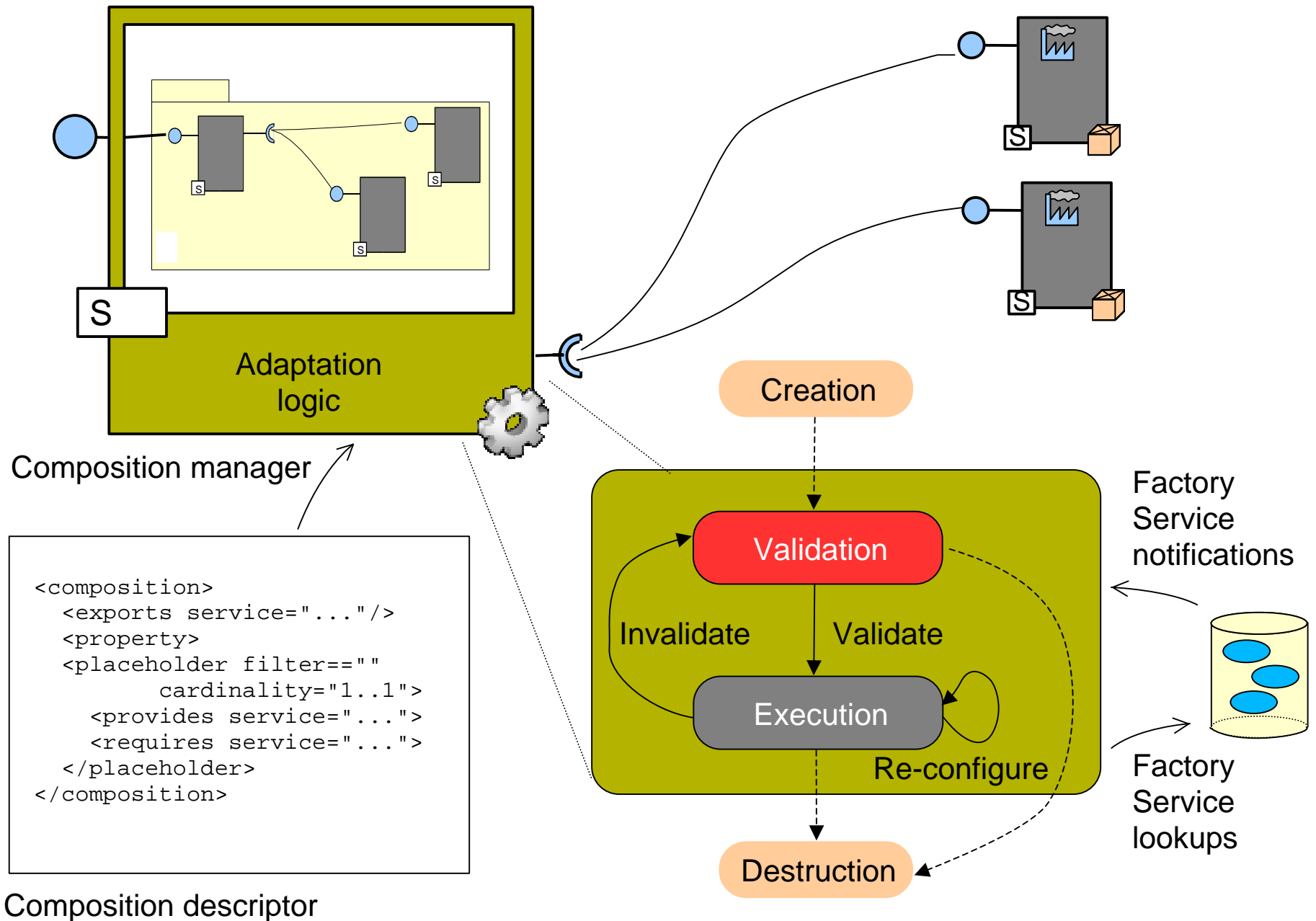
## An instance can be created inside a placeholder if

- Component fulfils the contract
- Additional provided services accepted
- Additional required services accepted if optional





# Composition-Level Management (7)





# Composition-Level Management (8)

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

**Summary**

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

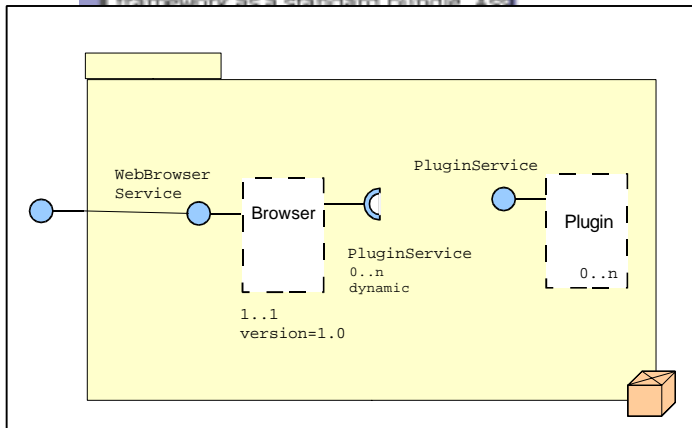
One of the most complex tasks faced by developers on this platform is writing assembly and adaptation logic in a dynamic and oriented environment. Inside such an environment, services exhibit **dynamic availability** (i.e., they arrive or depart at any time during execution), and applications must be 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 OSGi components can be built.
- [Browse the ServiceBinder's 1.1 framework API](#) (see [Version 1.0](#)).
- ["Automating Service Dependency Management in a Service-Oriented Component Model"](#), the ICSE CBSE6 Workshop, 2003.
- [ServiceBinder tutorial](#): This tutorial explains the basics of building applications with OSGi and the benefits of using the ServiceBinder.





# Composition-Level Management (8)

Launch Web Browser Web Browser

Web Browser  
URL: <http://www-adele.imag.fr/~cervante/servicebinder/> Back

## 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 by this platform is writing assembly and configuration in an object-oriented environment. Inside such an environment, objects (i.e., they arrive or depart at any time) are required to be capable of handling these situations. A ServiceBinder is responsible for writing both application and configuration logic in a general, complex, as it requires monitoring and managing these two types of logic end up intertwined and difficult.

The ServiceBinder solves this problem by automatically moving the bundles and moving it into an execution environment as a standard bundle. App...

Web Browser  
URL: <http://www-adele.imag.fr/BEANOME/papers/CervantesHallCBSE2003.pdf> Back

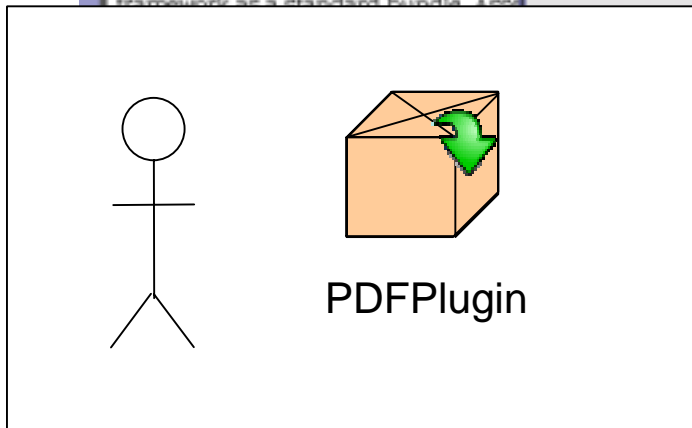
## Error Occurred

An error has occurred during the loading of the requested page.

### Error Details

```
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)
```

-adele.imag.fr/BEANOME/papers/CervantesHallCBSE2003.pdf





# Composition-Level Management (8)

The screenshot shows a web browser window with two tabs. The active tab displays a page titled "ServiceBinder" with the URL <http://www-adele.imag.fr/~cervante/servicebinder/>. The page content includes the title "ServiceBinder", a subtitle "Simplifying application development", authors "Humberto Cervantes, Richard S. Hall", and a "Summary" section. The summary text describes the ServiceBinder as a mechanism for automating service dependency management in a service-oriented environment, highlighting the complexity of handling dynamic availability of building blocks.

Overlaid on the browser is a PDF viewer window showing a 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. The abstract discusses the challenges of service dependency management in dynamic environments and the need for context-aware architectures. The introduction section is partially visible at the bottom of the PDF viewer.

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



# Validations and Project Status

---

## **Execution environment built on top of the Java-based OSGi services platform**

- Service registry
- Deployment mechanisms

## **Framework available for download**

- Open source project (<http://gravity.sourceforge.net>)
- Small execution environment (tested in a Zaurus PDA)
- Several research and commercial projects have created applications based on instance-level management



# Future Work (1)

---

## **Service and component description**

- Currently description is purely syntactic
- The concept of placeholder is too restrictive as it is too close to the component implementation, this could be improved to make it closer to a service description

## **More sophisticated rules for adaptation logic**

- To express aspects such as the temporalness of a binding
- To better deal with ambiguity

## **Lessen the impact of dynamic availability**

- State transfer
- UI changes



# Future Work (2)

## Plastic interfaces

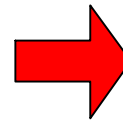
- Currently a simple approach is followed

The screenshot shows an IDE window titled "Gravity" with a menu bar "File Edit". The code editor contains the following Java code:

```
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;
}
```

Below the code editor are two buttons labeled "Buffer 0" and "Buffer 1". At the bottom, a file explorer shows the "Root dir" as "/home/rickhall/projects/win-projects/oscar/src/org" and a tree view with folders "org", "mortbay", "osgi", and "framework", and a file "AdminPermission.java".



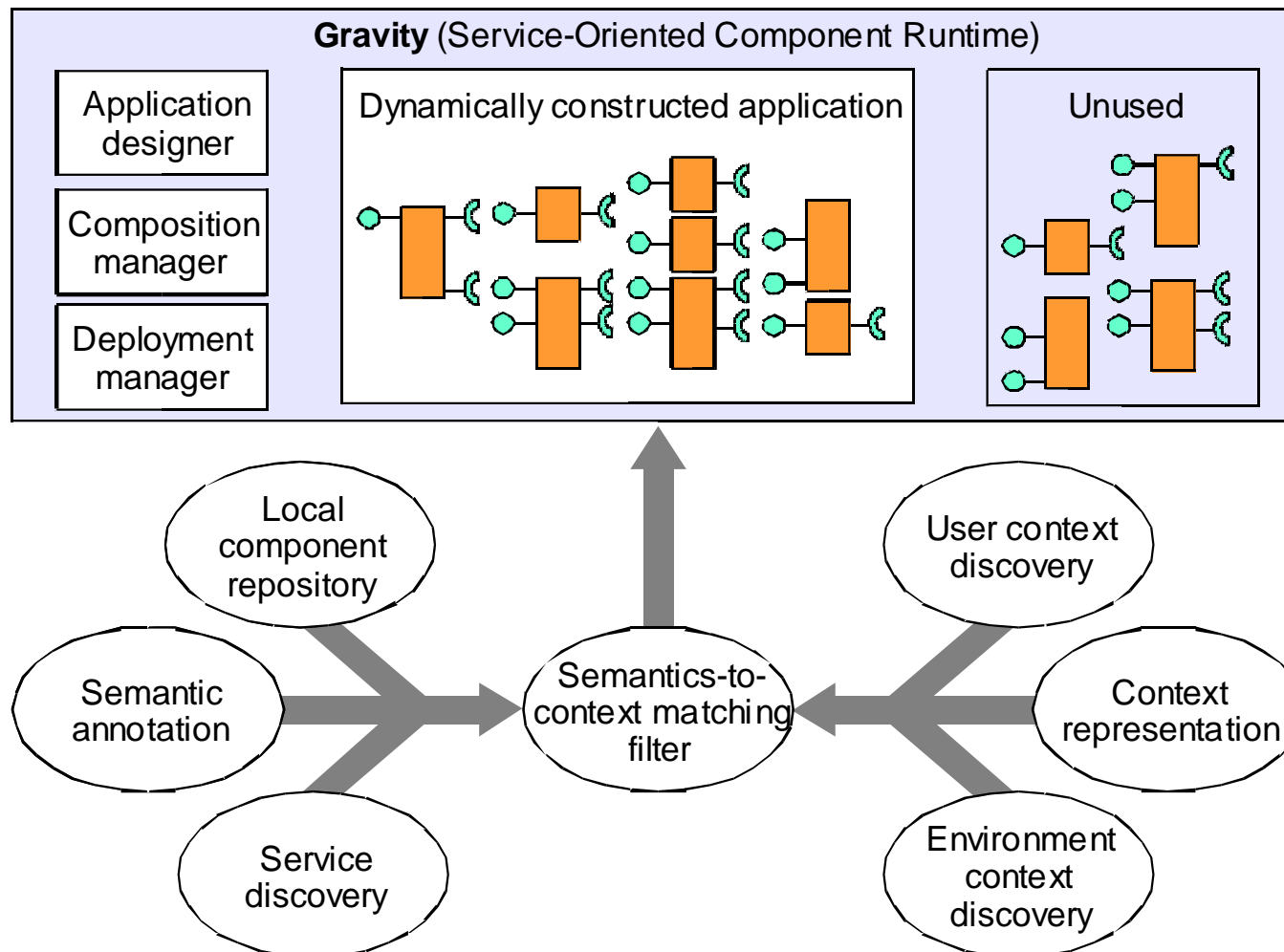
The screenshot shows the same IDE window "Gravity" with the same code as the previous window. However, the bottom section of the window is mostly empty, with a yellow warning icon (a person with a shovel) centered in the lower half. The "Buffer 0" and "Buffer 1" buttons are still present above the warning icon.



# Future Work (3)

## Study other causes for dynamic availability

- Contextual information could define availability







# Conclusions

---

## **Gravity follows a pragmatic approach to support dynamic availability in a component model**

- Adaptation logic is extracted from application logic
- Simple rules result in sophisticated behavior

## **Uncovered the major challenges of dynamic availability**

- Adapting to changes
- Dealing with ambiguity

## **Current results have been encouraging**

- Framework has been used in real projects
- Interest from the industry (OSGi Alliance)
- Many open areas for future research