



OSGi Alliance Community Event

**The Future Of Service-Oriented
Component Models for the OSGi
Framework**

**Clement Escoffier
Grenoble University**



Outline

- Service-oriented component model overview
- What do we get from a service-oriented component model ?
- iPOJO non-functional concern management
- Composing service-oriented components with iPOJO
- Conclusion



Nowadays ...

- Applications need to
 - Interoperate
 - Not all features are implemented by the same team / department / enterprise
 - Evolve quickly
 - Bug fixes, new features, etc.
 - Tackle environmental dynamism
 - Service dynamism
 - Implementation dynamism
- Need solutions to help architects & developers create applications living in this wild world
 - Provide a composition model and a development model supporting these constraints
 - Provide a framework supporting the execution of these applications

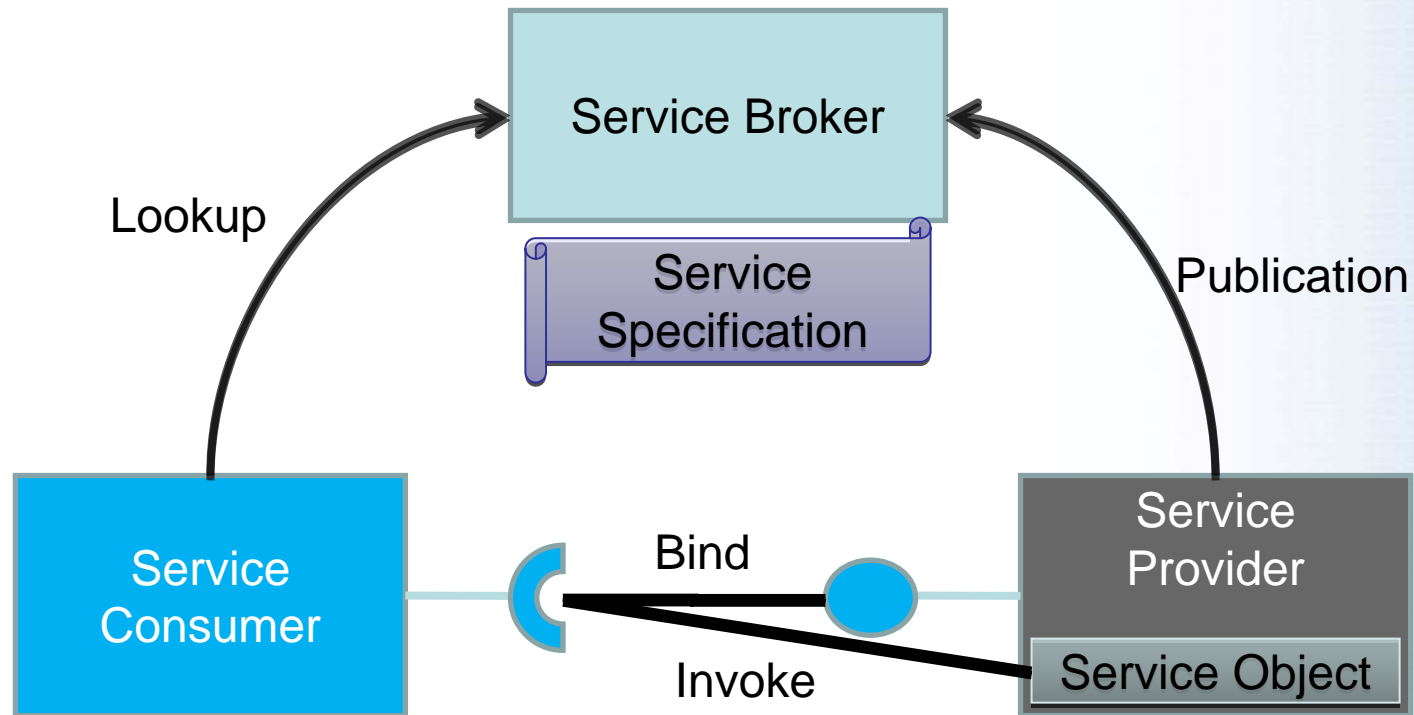


Service-Oriented Computing

- SOC : “New” computing paradigm
 - Service as fundamental element
 - Reduce dependencies among “software islands”
 - Separate evolution of different pieces
 - More flexible than monolithic approach
- SOC technologies today
 - Corba, Jini™, Web Services, OSGi™, UPnP™, DPWS™



Service-Oriented Computing Actors





Service-Oriented Computing Advantages

- SOC is a good challenger to tackle dynamic environments
 - Loose coupling – *Design by Contract*
 - Late binding – At runtime, on demand
 - Hides heterogeneity
- Adequate flexibility for dynamic applications



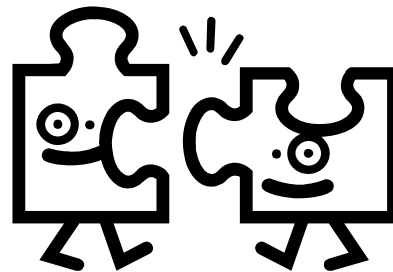
However ...

- Dynamism is difficult to manage
 - Service providers can arrive and disappear dynamically.
 - Service consumers cannot assume a needed service is available.
- Need a way to help architects design dynamic service-oriented applications
- Need an easy programming model to develop applications using dynamic services



Why not reuse component-based concepts ?

- Separation of concerns
 - Avoid mixing business code and non-functional concerns
- Avoid monolithic applications
 - An assembler uses (existing) components and composes them together (Architecture Description Language)





Component Concepts

- A component type
 - consistent piece of code
 - non-functional concerns configuration
 - defined interfaces (required and provided)
- A component instance is comprised of
 - Business logic code
 - A container that manages non-functional concerns
 - Binding, Lifecycle, Persistence, Security, Transaction ...
- An application is described in term of component instances and bindings among them

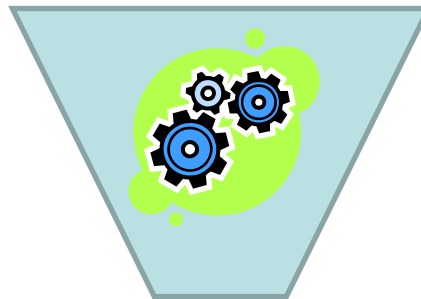




Service-oriented Component Models

Service Oriented
Computing

Component-Based
Software Engineering

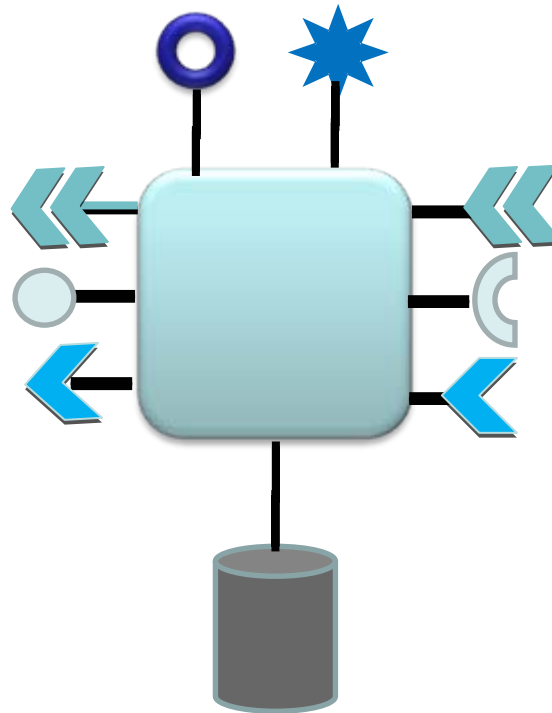


Service-Oriented
Component Model

- A service :
 - Provides some functionality
 - A service is described by a specification
- A service-oriented component
 - Provides services
 - Requires services
- Service dependencies are resolved at runtime
- Compositions are expressed in term of service specifications
- Applications are built by using available services



Design & Limits





Existing Models

- Service Binder 1.1 and 1.2 (Cervantes)
- Declarative Services (OSGi R4)
- Service Component Architecture (IBM)
- Dependency Manager (Offermans)
- Spring–OSGi (Adrian Colyer and all)
- iPOJO (Escoffier)



What are we getting from a service-oriented component model ?

- A simple and non-intrusive development model
- Service dynamics management
- Differentiation between component type / instance
- Other non-functional concerns management
 - Configuration, others connectors, persistence, transaction ...
- Extensibility, flexibility
- Composition/ADL
 - Hierarchic composition, isolation



- Extensible service-oriented component framework
 - Non-intrusive development model (POJO)
 - Manage service interactions
 - Extensibility mechanism
 - Composition mechanism
- An Apache Felix sub-project



Service Requirement Management

- Service dependencies
 - Aggregate, optional, filtered
 - Injection inside fields, via methods
 - *Nullable* object for unresolved optional dependencies

```
public class Foo {  
    private FooService m_foo;  
  
    public doSomething() {  
        ...  
        m_foo.foo();  
        ...  
    }  
}
```

```
<component className="..."  
    <requires field="m_foo"/>  
</component>
```



Service Publication

- Service publishing
 - Publish / revoke services
 - Provide service functionality
 - Manage object creation : singleton, OSGi™ service factory
 - Manage service properties
 - Fields : update the registration when the field value changed
 - Methods : be able to be notified of external value changes

```
public class FooImpl implement FooService {  
  
    public doSomething() {  
        ...  
    }  
}
```

```
<component className="...">  
    <provides/>  
</component>
```



Other Service Interactions

- Properties propagation
 - Enable / disable property propagation when reconfiguring an instance from the configuration admin
- Planned/in progress
 - Synchronization management
 - “Temporal” dependencies
 - Allow more fine-grained specification of dependencies
 - Requiring non-OSGi™ services - Bridges
 - Providing remote services (Web Service, DPWS, SCA...)



Service Management Comparison

- Very close to SCR service management, but...
 - Allows dynamic service properties
 - Not necessary to implement bind and unbind methods
- Spring-OSGi manages the dynamism differently
 - Unavailable Service Exception
- SCA does not support dynamism by default



Configuration & Management

- Instance properties
 - Allows injection of properties into instances
 - Enables creating two instances from the same type with different property values
- Creation of instances via factories
 - Configuration Admin support
 - Remotely (via JMX by using M-OSGi)
- Instance should be dynamically reconfigurable
 - Via Configuration Admin or via JMX



Others service-oriented component models

- SCR
 - Supports the Config Admin
 - Does not support JMX
- Spring-OSGi
 - Supports Config Admin & JMX reconfiguration
 - Does not support Configuration Admin instance management



Other non-functional management

- Connectors
 - Event binding



```
public class Receptor {  
    public receive(Event ev) {  
        ...  
    }  
}
```

```
<component className="...FooEventSubscriber">  
    <ev:subscriber topic="foo"  
        callback="receive" name="myEvent"/>  
</component>
```



Other non-functional management

- In progress
 - Persistence Management
 - Wire admin connectors
- Other service-oriented component models
 - Spring-OSGi provides JEE non-functional concerns management (Persistence, Transaction, JMS ...)



Extensibility

- Extensibility
 - Container is adaptable according to the context / application
 - Allows you to develop separate “handlers”
- Other service-oriented component models
 - Spring-OSGi is extensible (AOP)
 - SCA “wires” are extensible as implementation type



Introspection

- Allow reflecting on gateway architecture
 - Who uses who ?
 - Allows you to track unresolved dependencies, invalid handlers
- Others service-oriented component models
 - Service Binder provided an architecture view
 - Neither SCR nor Spring-OSGi provide this



Composing service-oriented components

- How to create new services or applications ?
 - Tackling the *wild world constraints*
- Service-oriented component model should provide composition mechanisms
 - Supporting service dynamism
 - Close to component-based composition



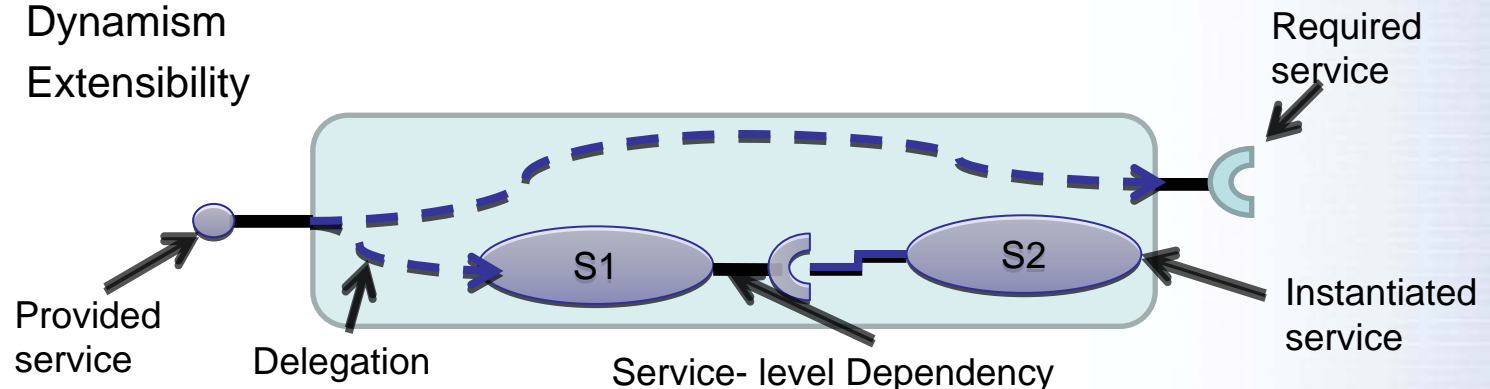
iPOJO Composition Model

- iPOJO composition merges service flexibility and component-based composition
 - Supports dynamism
- Composite components are expressed in term of
 - Contained services (and properties)
 - Required services (aggregate, optional, filter)
 - Provided services
 - Contained instances (and properties)



iPOJO Composition Model

- Contained services
 - Track service implementation and instantiate an instance inside the composite
 - Implementation can be composite too
- Composite can provide services
 - Abstract implementation
- Composite Properties
 - Isolation
 - Dynamism
 - Extensibility





Conclusion

- What do we get from future service-oriented component models ?
 - Simpler development model
 - Better management of service dynamism
 - Other non-functional concern management
 - Introspection ability
 - Extensibility
 - Composition Features
- Different contexts
 - Different requirements



Questions

