IPOJO: A FLEXIBLE SERVICE-ORIENTED COMPONENT MODEL FOR DYNAMIC SYSTEMS

Clement Escoffier
PhD Defense, December 3rd 2008
Université Joseph Fourier, Grenoble

Jury

Laurence Nigay, Présidente, Professeur, UJF, Grenoble
Alexander Wolf, Rapporteur, Professor, Imperial College, London
Michel Riveill, Rapporteur, Professeur, Polytech’Nice
François Exertier, Examinateur, Bull SAS
Philippe Lalanda, Directeur, Professeur, UJF, Grenoble
Richard Hall, Co-Directeur, Sun Microsystems
TWO PARALLEL EVOLUTIONS

Internet & The Web
- From static web to dynamic content
- Blur boundaries between desktop and Internet
- Towards Web 3.0

Ubiquitous computing
- Communicative & pervasive objects
- Exponential growth
- Seamless integration in the daily life
- Towards ambient intelligence
The “Crunch”

- Convergence between Internet and Ubiquitous Computing
  - Smart objects bring Internet closer to users

- Paves the road to new types of applications
  - Machine-to-machine
  - Home applications
THE “CRUNCH”

- A challenging convergence!
  - Requires facilities to design, develop, execute and manage.

- Emergence of new stringent requirements
  - Scalability
  - Security
  - Autonomy
  - Heterogeneity
  - Evolution
THE “CRUNCH”

- A challenging convergence!
  - Requires facilities to design, develop, execute and manage.

- Emergence of new stringent requirements
  - Scalability
  - Security
  - Autonomy
  - Heterogeneity
  - Evolution
OUTLINE

- Being Dynamic, Why, What and How?
- Service-Oriented Computing & Dynamism
- Problematic & Objectives
- iPOJO: Principles & Concepts
- Dynamism in Atomic & Composite Components
- Implementation & Validation
- Conclusion & Perspectives
BEING DYNAMIC
Why, What and How?
WHAT DOES “DYNAMIC” MEAN?
INTERNAL EVOLUTION
What Does “Dynamic” Mean?
Environmental Changes
What Does “Dynamic” Mean? Contextual Changes
WHAT IS A “DYNAMIC APPLICATION”?

- Dealing with dynamism impacts the architecture
  - Adding, removing, updating components
  - Modifying connectors

A dynamic application is an adaptable application supporting the modification of its architecture during its execution

- Flexible, efficient .... complex to design, develop, execute and manage!
Guarantying application consistency is complex
- Notions of quiescence / tranquility states
EXISTING APPROACHES

- **Ad-hoc approaches**
  - Context-aware applications, product-lines, autonomic,
  - Hard to generalize

- **Component models supporting dynamic reconfiguration**
  - SOFA/DCUP, OpenRec, ...
  - Focused on a given type of dynamism, lack of flexibility

- **Extended architecture description languages**
  - Darwin, Dynamic Wright, C2ADEL, ...
  - Big gap between such languages and execution frameworks
SYNTHESIS

- Dynamism is today needed but extremely complex to manage

- Existing solutions are limited
  - Require a lot of design and development effort
    - State management, synchronization, ...
  - Do not always support the different types of dynamism
    - Constrained to specific domains
    - *Ad-hoc* mechanisms
SERVICE-ORIENTED COMPUTING & DYNAMISM
Towards Dynamic Extended Service-Oriented Architecture
SERVICE-ORIENTED COMPUTING (SOC)  
“PUBLISH-FIND-BIND”

- Service Broker
- Discovery
- Publication
- Service Specification
- Service Consumer
- Binding
- Required Service
- Provided Service
- Service Provider
MAIN CHARACTERISTICS

- Loose-coupling: only the specification is shared
- Late-binding: on-demand binding
- Substitutability: a provider can be replaced

SOC interactions can happen at runtime: Consumers can adapt themselves to service dynamism
**Dynamic SOC**

**Service Provider Arrival**

1) Publication

2) Notification

3) Bind

**Service Provider Departure**

1) Withdrawal

2) Notification

3) Unbind
SERVICE-ORIENTED ARCHITECTURE (SOA)

- SOA, a technical environment made of:
  - A service specification format
  - A publication/query mechanism
  - An interaction protocol

- Examples
  - WS:  <WSDL, UDDi, SOAP>
  - OSGi™:  <Interface Java, Service Registry, Direct>  Dyn.
EXTENDED SOA (FROM PAPAZOLOU)

- Monitoring & Management
  - Metrics
  - Evolution
  - State
- Non-functional properties QoS
- Composition
  - Conformance
  - Transactions
  - Coordination
- Foundation (SOA)
  - Discovery
  - Publication
  - Interface, Behavior, Capability
  - Selection
  - Binding
**Dynamic Extended SOA (Proposed)**

- **Foundation (Dynamic SOA)**
  - Discovery
  - Interface, Behavior, Capability
  - Notification
  - Registration
  - Selection
  - Binding

- **Dynamic Composition**
  - Dynamic Composition
  - Coordination
  - Transactions

- **Non-functional properties QoS**
  - Evolution
  - Reconfiguration
  - Introspection
  - State

- **Monitoring & Management**
  - Monitoring & Management
  - Conformance
  - Transactions

- **Layered Architecture**
  - Evolution
  - Reconfiguration
  - Introspection
  - State
  - Conformance
  - Transactions
  - Monitoring & Management
SUMMARY: SOC, SOA, EXTENDED SOA AND FRIENDS...

- SOC
  - Paradigm based on services promoting loose-coupling

- SOA
  - Set of technologies allowing the development and execution of applications following SOC principles

- Dynamic SOC
  - Paradigm based on the SOC but adding primitives to support dynamism

- Dynamic SOA
  - Set of technologies allowing the development and execution of dynamic applications following dynamic SOC principles

- Extended SOA
  - Set of technologies allowing the development, composition, management and execution of applications following SOC principles.
  - Is based on a SOA

- Dynamic Extended SOA
  - Set of technologies allowing the development, composition, management and execution of applications following dynamic SOC principles.
  - Is based on a dynamic SOA
SERVICE-ORIENTED COMPONENT MODELS (SOCM)

- SOCM infuses SOC dynamic principles inside component models

- Principles (Cervantes, Hall):
  - A service is a specified functionality
  - A component instance provides and requires services
  - Bindings between instances follow the SOC dynamic interaction pattern
  - Compositions are described in terms of specifications
  - Service specifications form the basis for substitution
Are SOCM Dynamic Extended SOA?

- Yes, SOCM:
  - are based on a dynamic SOA
  - provide composition mechanisms
  - provide monitoring and administration mechanisms

- But … No! Existing SOCM don’t provide all capabilities
  - Focus on the development model simplification
    - SCR, Spring-DM
  - Compositions are generally not supported (or are static)
    - Apache Tuscany (SCA), Spring-DM
  - Administration and monitoring funct. are very limited

But it is a promising path
PROBLEMATIC AND OBJECTIVES
# Dynamic Applications

## Current State

<table>
<thead>
<tr>
<th>Approaches</th>
<th>Pros</th>
<th>Cons</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Component Models</strong></td>
<td>• Structural composition</td>
<td>• Lack of flexibility</td>
</tr>
<tr>
<td></td>
<td>• Simplify the dev. model</td>
<td>• Difficulties to manage contextual and env. dynamism</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Dynamic Service Oriented Architecture</strong></td>
<td>• Loose-coupling</td>
<td>• No architectural view</td>
</tr>
<tr>
<td></td>
<td>• Late binding</td>
<td>• No admin. features</td>
</tr>
<tr>
<td></td>
<td>• Substitutability</td>
<td>• Development model difficult to control</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Service-Oriented Component Models (Dynamic Extended SOA)</strong></td>
<td>• Structural composition</td>
<td>• Composition rarely provided or static</td>
</tr>
<tr>
<td></td>
<td>• Simplify the dev. model</td>
<td>• Has generally an impact on the application code</td>
</tr>
<tr>
<td></td>
<td>• Handle dynamism</td>
<td></td>
</tr>
</tbody>
</table>
**Goal: A Service-Oriented Component Model**

- Providing a component model supporting dynamism and an associated execution framework
- Defining a service-oriented architecture providing features to manage dynamism and structural compositions
- Proposing an “as simple as possible” development model
- Defining a composition language
- Providing introspection and reconfiguration capabilities
- Providing an extensibility mechanism to adapt the component model, and the runtime
IPOJO, Our Proposal

- A service-oriented component model
  - Supporting structural compositions
    - Hierarchical
  - Built applications are natively dynamic
  - Extensible (implemented with an open container)

- Key concepts
  - Service implementations and instances
  - A service specification model
  - A service dependency model
  - Service context
COMPONENT TYPES & INSTANCES

- Component Types
  - Implementations (code!)
  - Describe provided and required services
  - Supports updates

- Instances
  - “Living” entities
  - Requires and Provides services
  - Introspectable
SERVICE SPECIFICATION

- A service is described with
  - Java interface
  - Properties (open set)
  - State
  - Service dependencies

- Designed to support structural composition
  - Applications are designed using composable services specification
A Rich and Flexible Dependency Model

- Two levels of dependencies
  - Service-level
  - Implementation-level

- A dependency targets a service specification
  - scalar or aggregate
  - optional or mandatory
  - can be filtered and/or sorted
  - binding policies
    - Dynamic, Static, Dynamic-Priority

- Properties
  - Reconfigurable, *Introspectable*
SERVICE CONTEXT

- Hierarchical structural Service Composition

- Support service isolation
  - Notion of service contexts
    - Equivalent to isolated dynamic SOAs
  - Each composition has its own service context
    - Isolates instances created in the composition
IPOJO & EXTENDED DYNAMIC SOA

- Supports Evolution, Introspection, Reconfiguration
- Provides mechanism to execute dynamic hierarchical structural service composition
  - Service Specification model
  - Dependency Model
- Provides a hierarchical dynamic SOA
  - Service Context
  - Service Implementation/Service Instance
DYNAMISM MANAGEMENT IN ATOMIC & COMPOSITE COMPONENTS
ATOMIC COMPONENT

Atomic components deal with the following requirements

- A simple development model,
- Hiding dynamism,
- Managing state

Characteristics

- Centered on the notion of service component
  - With required and provided services
  - Partial architectural vision
- It is a component type with a concrete implementation, supporting configurations
Example of Atomic Component Description

```java
@Component
@Provides
public class AlarmServiceImpl implements AlarmService {
    @Requires
    private MessageSender m_sender;
    public void sendAlarm(String message) {
        System.out.println(m_sender.send(message));
    }
}
```

AlarmService
Example Of Atomic Component Service Departure Management

```java
@Component
@Provides
public class AlarmServiceImpl implements AlarmService {
    @Requires
    private MessageSender m_sender;
    public void sendAlarm(String message) {
        System.out.println(m_sender.send(message));
    }
}
```

AlarmService
EXAMPLE OF_ATOMIC COMPONENT SERVICE ARRIVAL MANAGEMENT

@Component
@Provides
public class AlarmServiceImpl implements AlarmService {
    @Requires
    private MessageSender m_sender;
    public void sendAlarm(String message) {
        System.out.println(m_sender.send(message));
    }
}
COMPOSITE COMPONENT

- An Architecture Description Language defined in terms of:
  - Required Service Specifications
    - Instantiated and Imported
  - Provided Service Specifications
    - Exported and *Implemented*
  - Component Types

- Characteristics
  - Application concept and vertical composition
  - Implementation evolution and substitution
  - Context-awareness
COMPOSITE COMPONENT DESCRIPTION

Provided Service

Instance of Component Type

Instance of service Implementation

Delegation Scheme

Imported Service
COMPOSITE COMPONENT EXAMPELE

```xml
<composite name="Editor1">
  <subservice action="instantiate"
    specification="...Editor"/>
  <subservice action="instantiate"
    specification="... Plugin"
    aggregate="true"/>
  <subservice action="import"
    specification="...Printer"
    optional="true"/>
</composite>
```
COMPOSITE COMPONENT
CONTEXT-AWARENESS EXAMPLE

<composite name="Editor2">
<subservice action="instantiate"
    specification="... Plugin" aggregate="true"
    filter="(type=${my.type}) "
    context-source="local:editor" />
<subservice action="instantiate"
    specification="...Editor"/>
<subservice action="import"
    specification="...Printer" optional="true"/>
</composite>
COMPOSITE COMPONENT
CONTEXT-AWARENESS EXAMPLE

<composite name="Editor2">
<subservice action="instantiate"
    specification="... Plugin" aggregate="true"
    filter="(type=${my.type}) "
    context-source= "local:editor" />
<subservice action="instantiate"
    specification="...Editor" />
<subservice action="import"
    specification="...Printer" optional="true"/>
</composite>
**Composite Component Context-Awareness Example**

- The printer can also become context-aware
  - Select the of the closest printer

```xml
<composite name="Editor3">
  <subservice action="instantiate"
    specification="... Plugin" aggregate="true"
    filter="(type=${my.type})"
    context-source="local:editor" />
  <subservice action="instantiate"
    specification="...Editor"/>
  <subservice action="import"
    specification="...Printer" optional="true"
    context-source="global:location-source"
    filter="(&(printer.location=${current.location})(duplex=true))"
  />
</composite>
```

- To get the closest printer, the composition uses a global context-source tracking the user location
OTHER FEATURES: INTROSPECTION, RECONFIGURATION & EXTENSIBILITY

- System introspection for monitoring purposes
- System reconfiguration
- Supports extensions
SYNTHESIS

- Atomic Components provide a simple dev. model
  - Hiding dynamism
  - Hiding service-based interactions
  - Hiding synchronization

- Composites provide an ADL for dynamic applications
  - Based on services
  - Supporting evolution dynamism, environmental changes and context changes

- Noteworthy features
  - Introspection, reconfiguration, extensions support
IMPLEMENTATION & VALIDATION
IMPLEMENTATION

- iPOJO implementation main features
  - *Bytecode* manipulation
  - Extensible through *Handlers*
    - Handlers are iPOJO instances
    - Natively support dynamism
  - Heavy use of threads and synchronization constructions
  - On top of OSGi R4.0
Validation Benchmark

- Impact on the code size
  - According to the application, iPOJO can drastically reduce the number of line of code

- Several benchmarks were executed
  - Startup time of large applications (vs. OSGi)
    - Facing the “Event Storm”
    - OSGi: 512 687 ms / iPOJO: 491 543 ms
  - Service Access
    - Analyze service injection against other injection frameworks
INJECTION BENCHMARK RESULTS
RESIDENTIAL GATEWAYS EXAMPLE

- OSGi/iPOJO framework is used to develop residential gateways.

- Requirements:
  - Dynamism management
  - Extensibility
  - Composition and Isolation
JAVA EE SERVER EXAMPLE

- iPOJO is used in the JOnAS Java EE server
- Requirements
  - Dynamism management
  - Non-intrusive development model
Main Contributions

- iPOJO proposes a new way to design, develop and execute dynamic applications
- A model and an associated runtime
- Provides a simple development model
- Provides a hierarchical composition language
- Provides introspection, reconfiguration and extensibility mechanisms
AVAILABILITY

- iPOJO is hosted on Apache Felix
  - Every described feature is implemented!

- Additional provided tools
  - Integration in the build process
    - Ant, Maven
  - A command dumping instance architecture data
  - A test framework (based on Junit)
PERSPECTIVES

- Apply iPOJO principles on different technologies
  - Principles can also be used on the top of other technologies than OSGi™
  - However, rare are the frameworks providing the required underlying functionalities

- Deployment support
  - How to ease the deployment of dynamic applications?

- Context-Aware and Autonomic Applications
  - iPOJO can be used to execute context-aware and autonomic applications
  - What are the missing features?
PERSPECTIVES

- Domain-driven application servers
  - How to provide an ADL, an IDE and an execution framework for a specific domain
  - iPOJO extensibility mechanisms can be applied to solve such problems.
  - Ongoing ...

Specific Composition Language

Specialized IDE

Execution Environment

Application

Handlers

Technical Services

iPOJO
APPENDIX A
INTERCEPTION & INJECTION FRAMEWORK

Diagram showing the relationship between Injection, Container, Implementation class instance, Injection & Interception Layer, and Interception.
APPENDIX A
INTERCEPTION & INJECTION FRAMEWORK

public class ClientImpl implements Pojo {

    private Service __getm_service() {
        if(!__Fm_service) return m_service;
        else return (Service)__IM.onGet(this, "m_service");
    }

    private void __setm_service(Service service) {
        if(!__Fm_service) { m_service = service; }
        else { __IM.onSet(this, "m_service", service); }
    }

    public ClientImpl() { this(null); }
    private ClientImpl(InstanceManager _manager) { __setInstanceManager(_manager); }

    public void doSomething() {
        if(!__MdoSomething) {
            __doSomething(); return;
        }
        try{
            __IM.onEntry(this, "doSomething", new Object[0]);
            __doSomething();
            __IM.onExit(this, "doSomething", null);
        } catch (Throwable throwable) {
            __IM.onError(this, "doSomething", throwable);
            throw throwable;
        }
    }

    private void __doSomething() { System.out.println(__getm_service().getMessage()); }

    private void __setInstanceManager(InstanceManager instanceManager) { ... }
    public ComponentInstance getComponentInstance() { return __IM; }

    private InstanceManager __IM;
    private boolean __Fm_service;
    private Service m_service;
    private boolean __MdoSomething;
}
## Appendix B

### Lines Of Code

<table>
<thead>
<tr>
<th>Core</th>
<th>Projects</th>
<th>LOC</th>
<th>Test LOC</th>
</tr>
</thead>
<tbody>
<tr>
<td>Execution Framework</td>
<td>7500</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Manipulator</td>
<td>2350</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Metadata</td>
<td>242</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Annotations</td>
<td>105</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Core</td>
<td></td>
<td></td>
<td>30000</td>
</tr>
<tr>
<td>Composition Model</td>
<td>Composite</td>
<td>2900</td>
<td>8000</td>
</tr>
<tr>
<td>Tools</td>
<td>“arch”</td>
<td>130</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Maven plugin</td>
<td>70</td>
<td>8500</td>
</tr>
<tr>
<td></td>
<td>Ant Taks</td>
<td>80</td>
<td></td>
</tr>
<tr>
<td></td>
<td>OBR support</td>
<td>2400</td>
<td></td>
</tr>
<tr>
<td>Tools</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>External handlers</td>
<td>Event Admin</td>
<td>300</td>
<td></td>
</tr>
<tr>
<td>Temporal Dependencies</td>
<td>250</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Extension &amp; Whiteboard</td>
<td>330</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Administration</td>
<td>670</td>
<td></td>
<td></td>
</tr>
<tr>
<td>External handlers</td>
<td></td>
<td></td>
<td>9500</td>
</tr>
</tbody>
</table>