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
Francois Exertier,	Examinateur,	Bull SAS
Philippe Lalanda,	Directeur,	Professeur, UJF, Grenoble
Richard Hall,	Co-Directeur,	Sun Microsystems

Two Parallel Evolutions

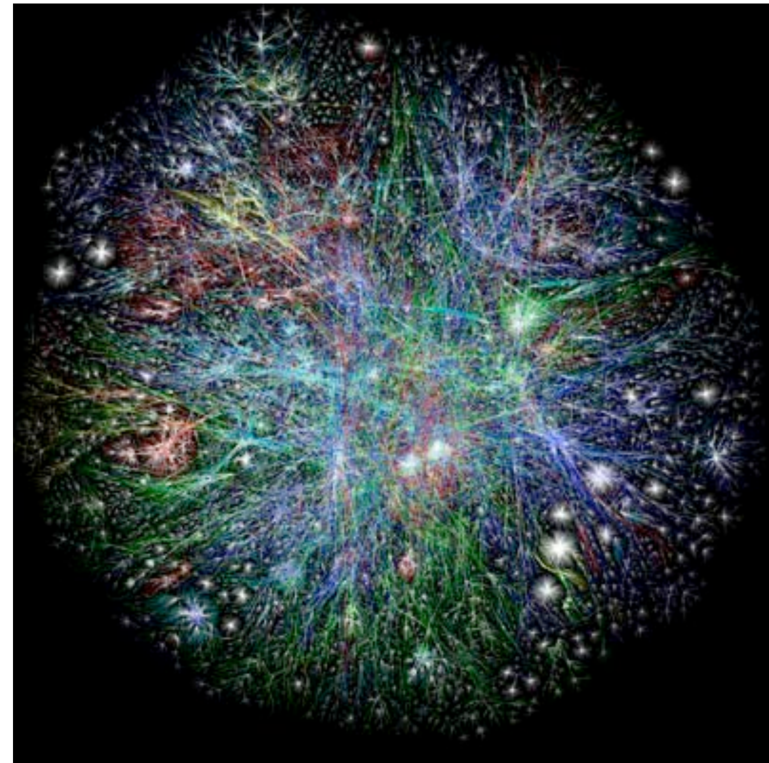
- Internet & The Web
  - From static web to dynamic content
  - Blur boundaries between desktop and Internet
  - Towards Web 3.0

- 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

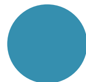
The "Crunch"

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

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

Outline

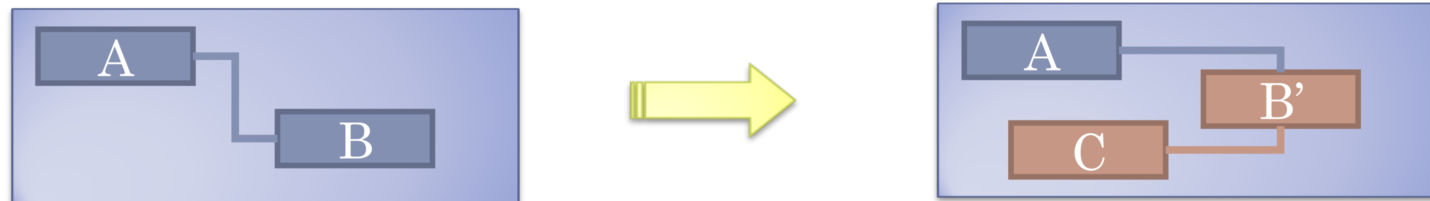
- Being Dynamic, Why, What and How ?
- Service-Oriented Computing & Dynamism
- Problematic & Objectives
- iPOJO: Principles & Concepts
- Dynamism in Atomic & Composite Components
- Implementation & Validation
- Conclusion & Perspectives



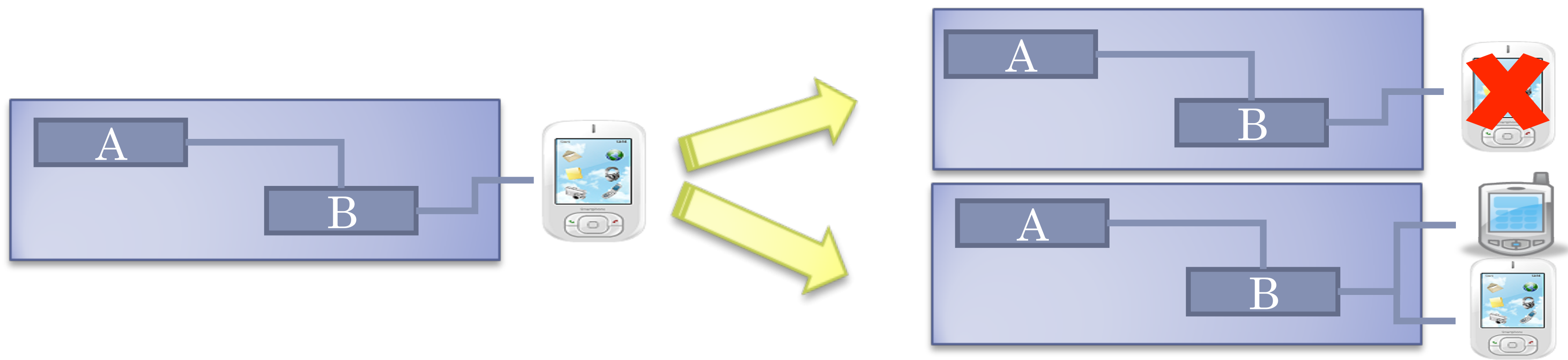
Being Dynamic

Why, What and How ?

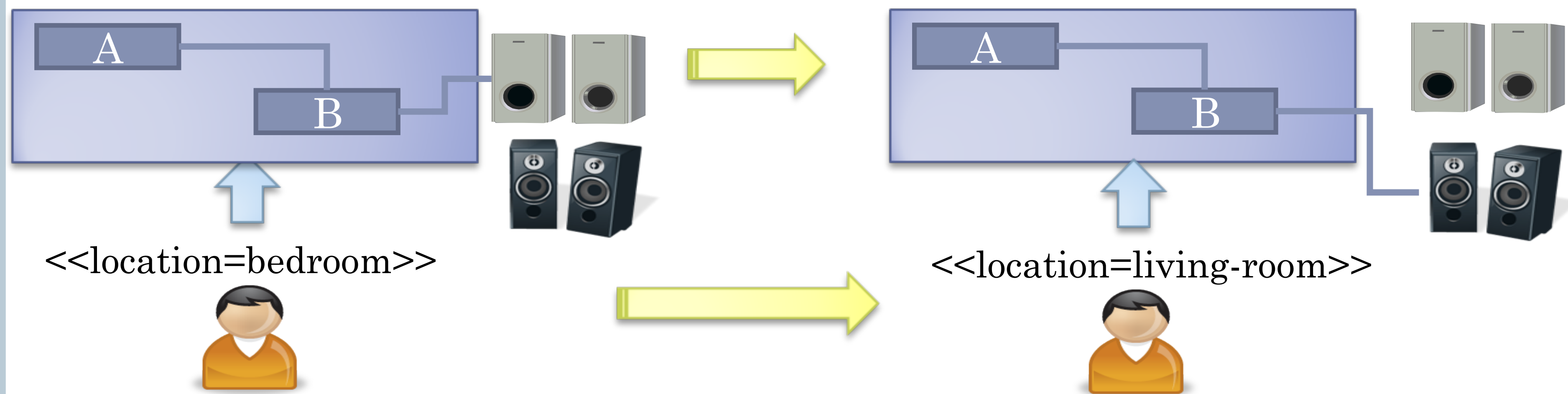
What Does "Dynamic" Mean?
Internal Evolution



WHAT DOES "DYNAMIC" MEAN?
ENVIRONMENTAL CHANGES



What Does "Dynamic" Mean?
Contextual Changes



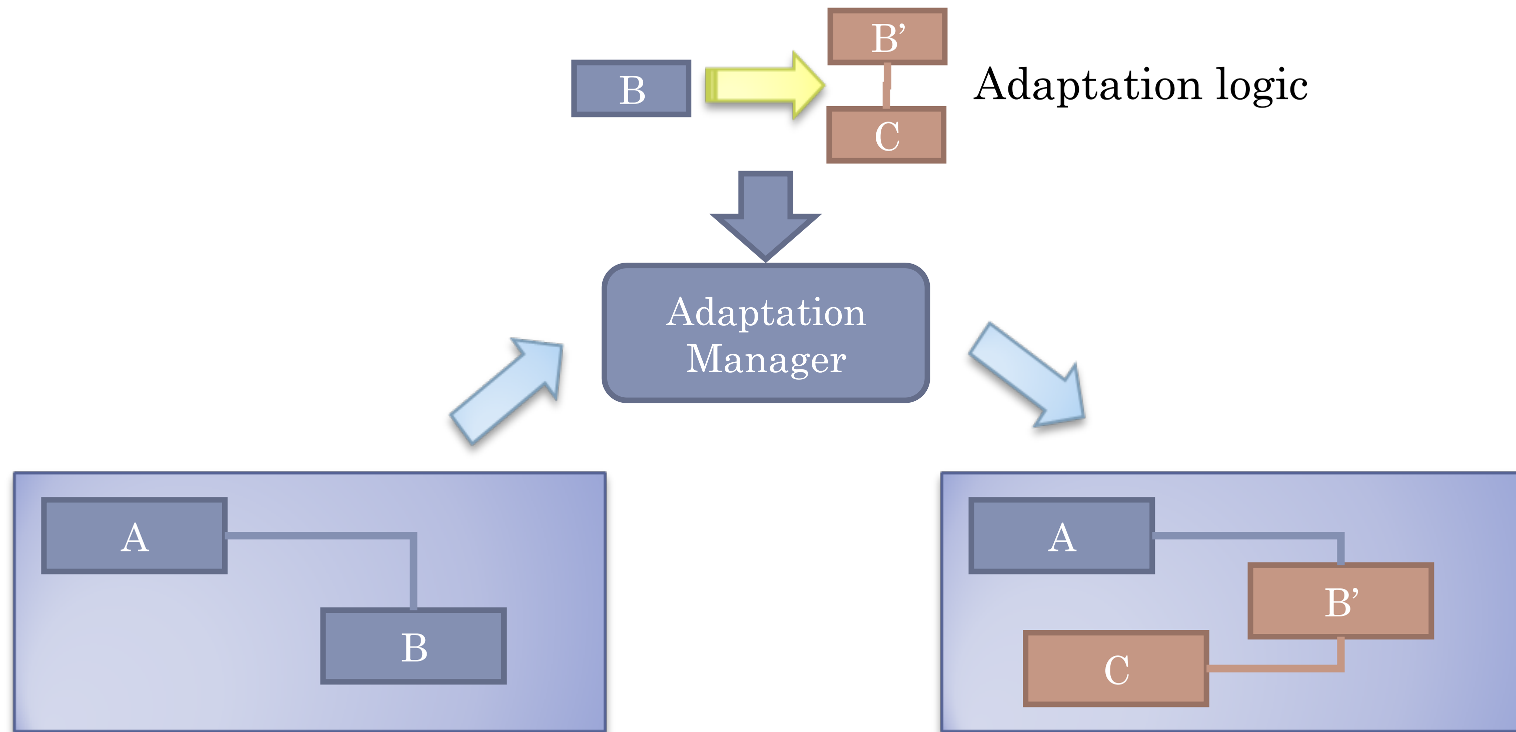
WHAT IS A "DYNAMIC APPLICATION" ?

- 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!

How Is An Application "Adapted"?



- 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

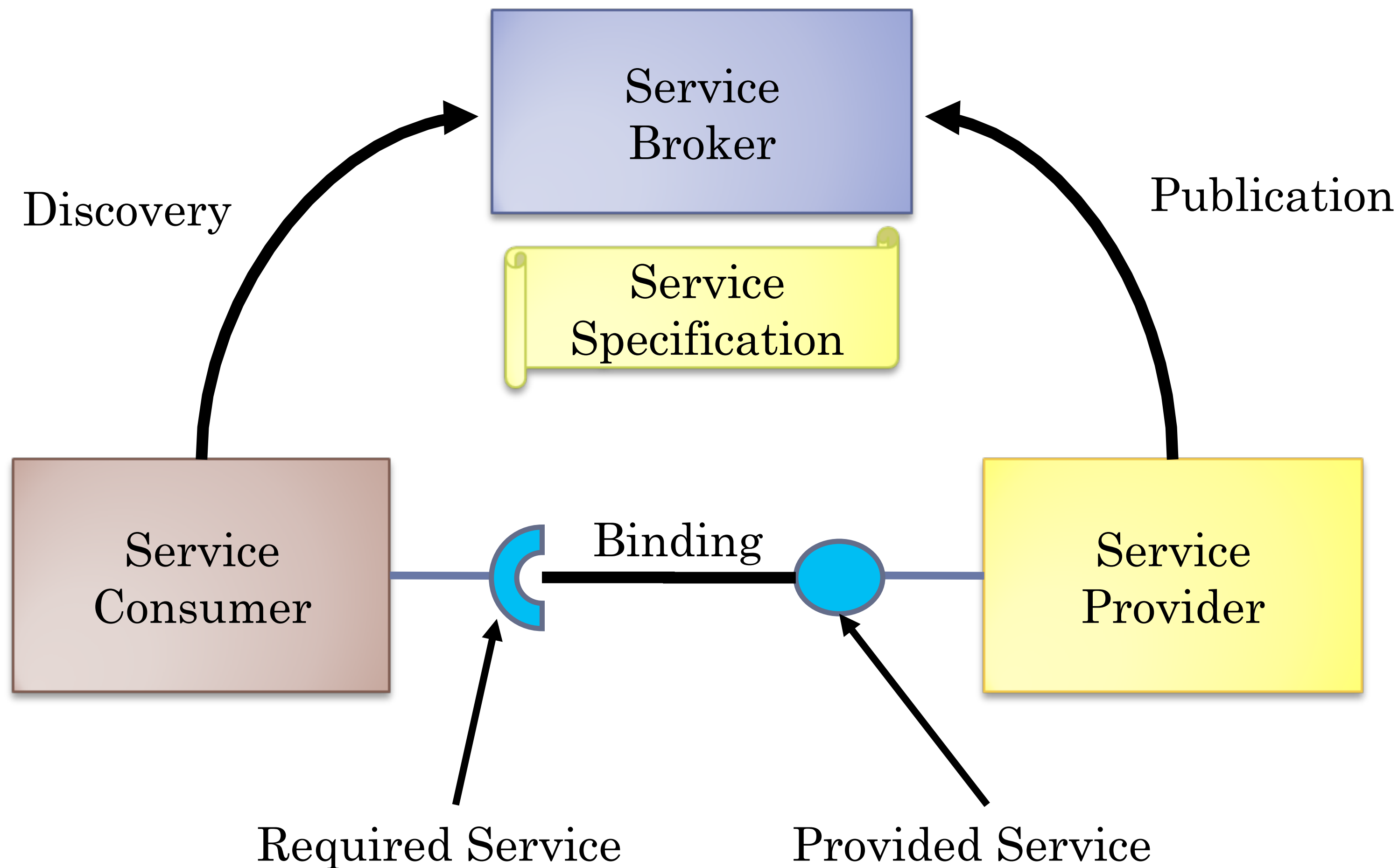
- Dynamism is today needed but extremely complex to manage

- 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"

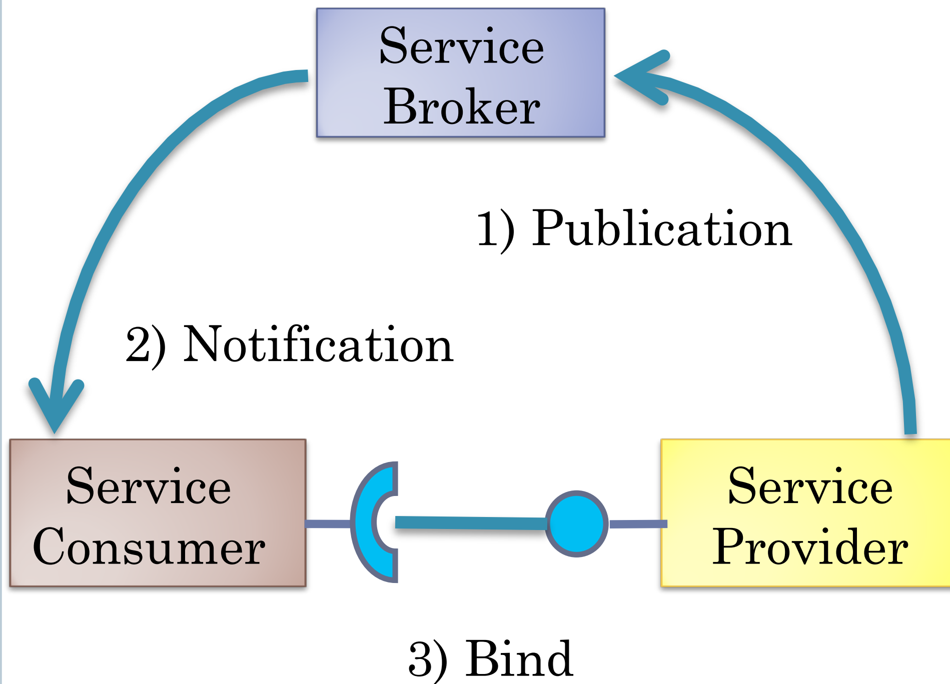


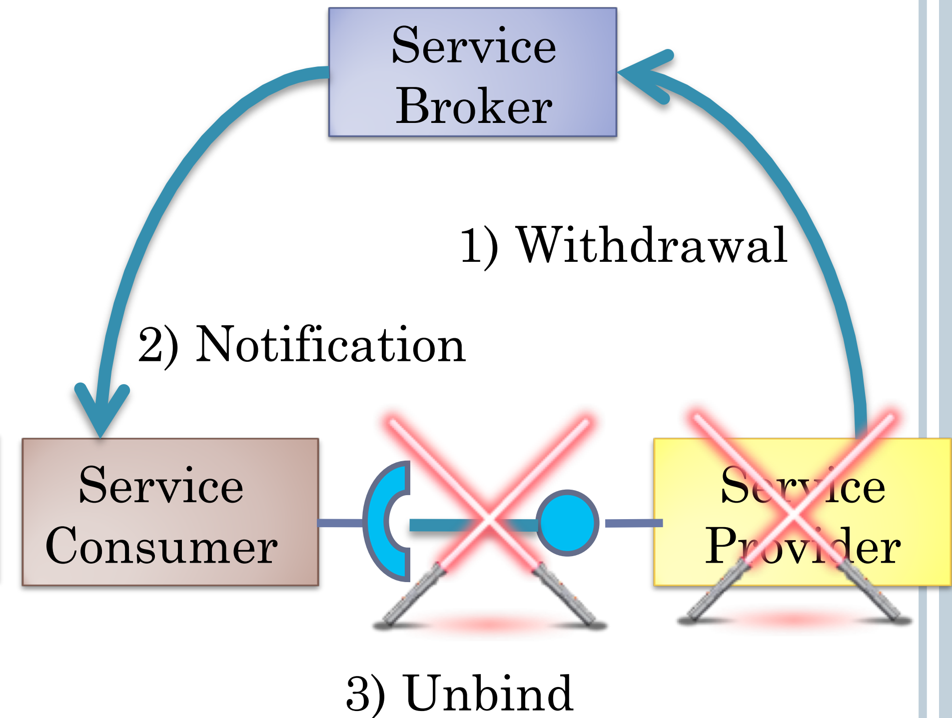
MAIN CHARACTERISTICS

- 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



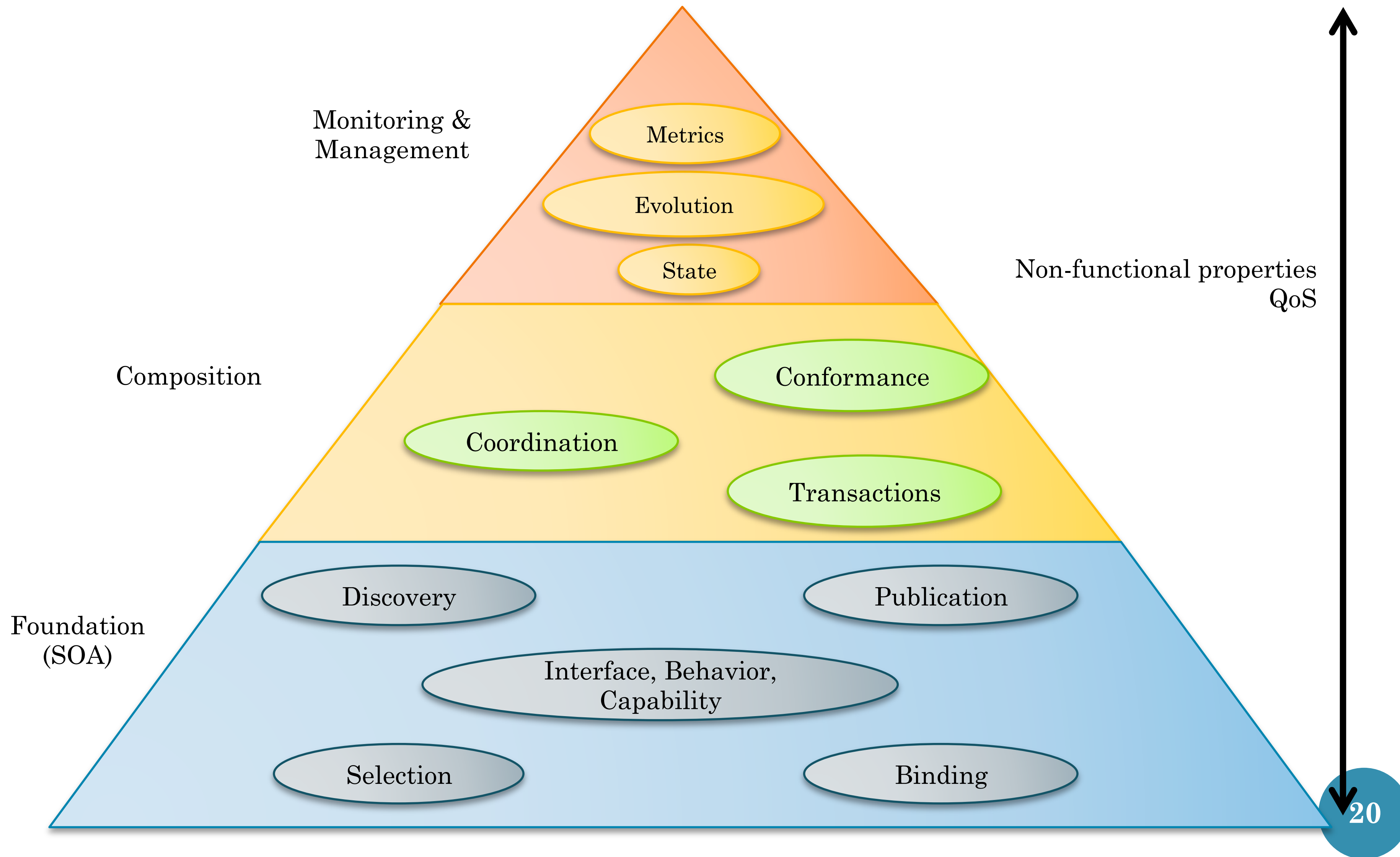


SERVICE-ORIENTED ARCHITECTURE (SOA)

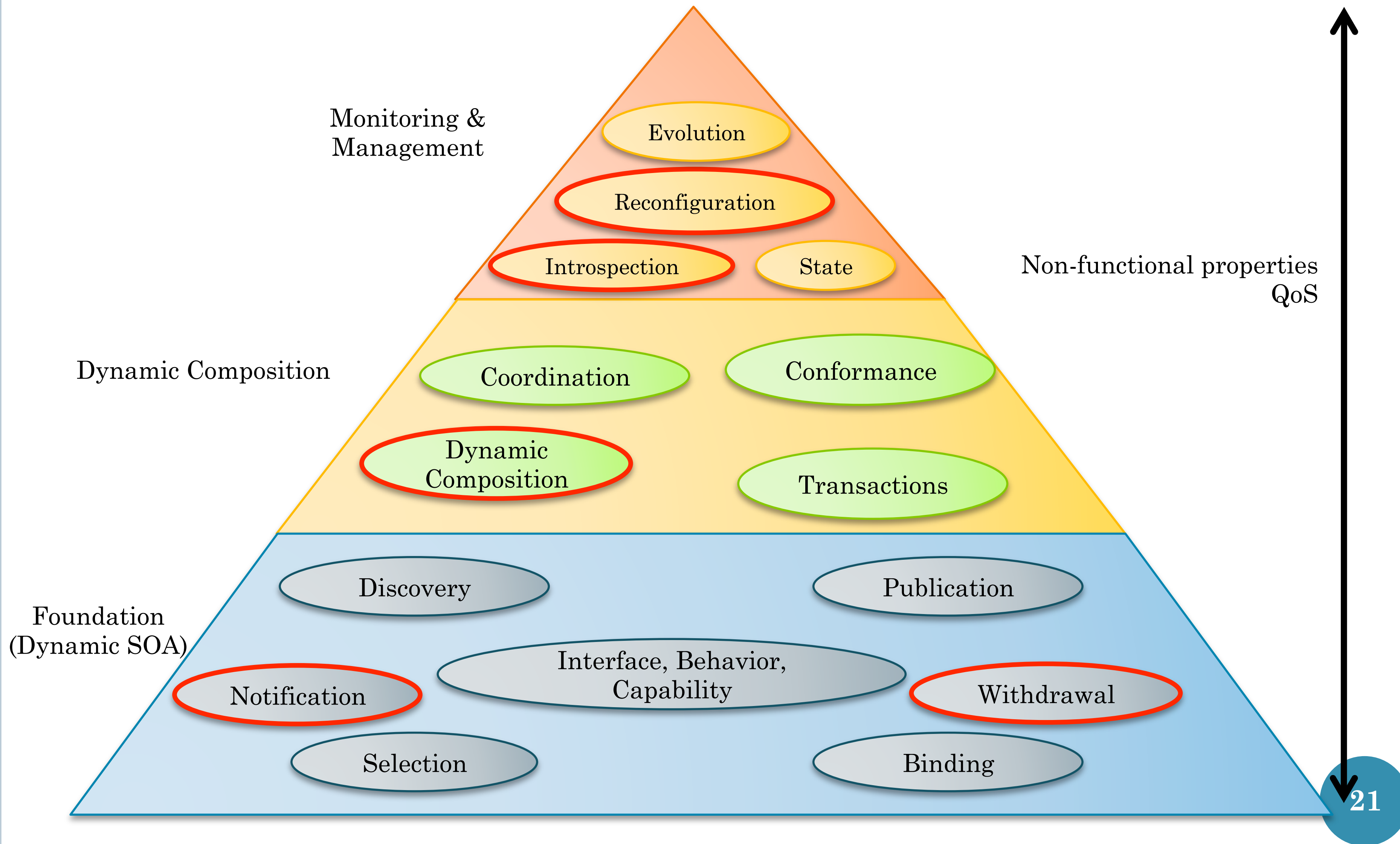
- SOA, a technical environment made of:
  - A service specification format
  - A publication/query mechanism
  - An interaction protocol

- Examples
  - WS: <WSDL, UDDi, SOAP>
  - Corba: <IDL, Corba Trading Service, IIOP> **Dyn.**
  - Jini: <Interface Java, Discovery Service, RMI> **Dyn.**
  - OSGi™: <Interface Java, Service Registry, Direct> **Dyn.**

Extended SOA (from Papazoglou)



Dynamic Extended SOA (Proposed)



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

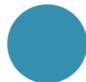
- Principles (Cervantes, Hall):
  - A service is a specified functionality
  - A component instance provides and requires services
  - Bindings between instances follow the SOC dynamic interaction pattern
  - Compositions are described in terms of specifications
  - Service specifications form the basis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

Are SOCM Dynamic Extended SOA?

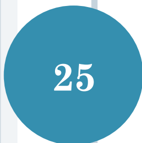
- Yes, SOCM:
  - are based on a dynamic SOA
  - provide composition mechanisms
  - provide monitoring and administration mechanisms

- 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

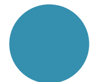


DYNAMIC APPLICATIONS
CURRENT STATE

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

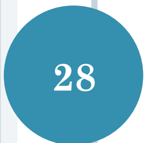
Goal: A Service-Oriented Component Model

- 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



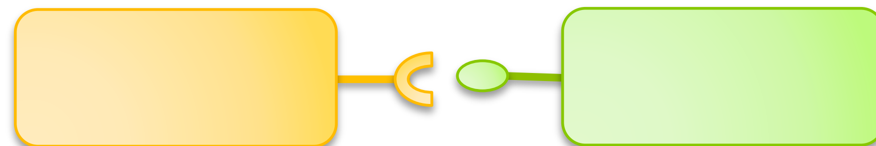
iPOJO

Principles & Concepts



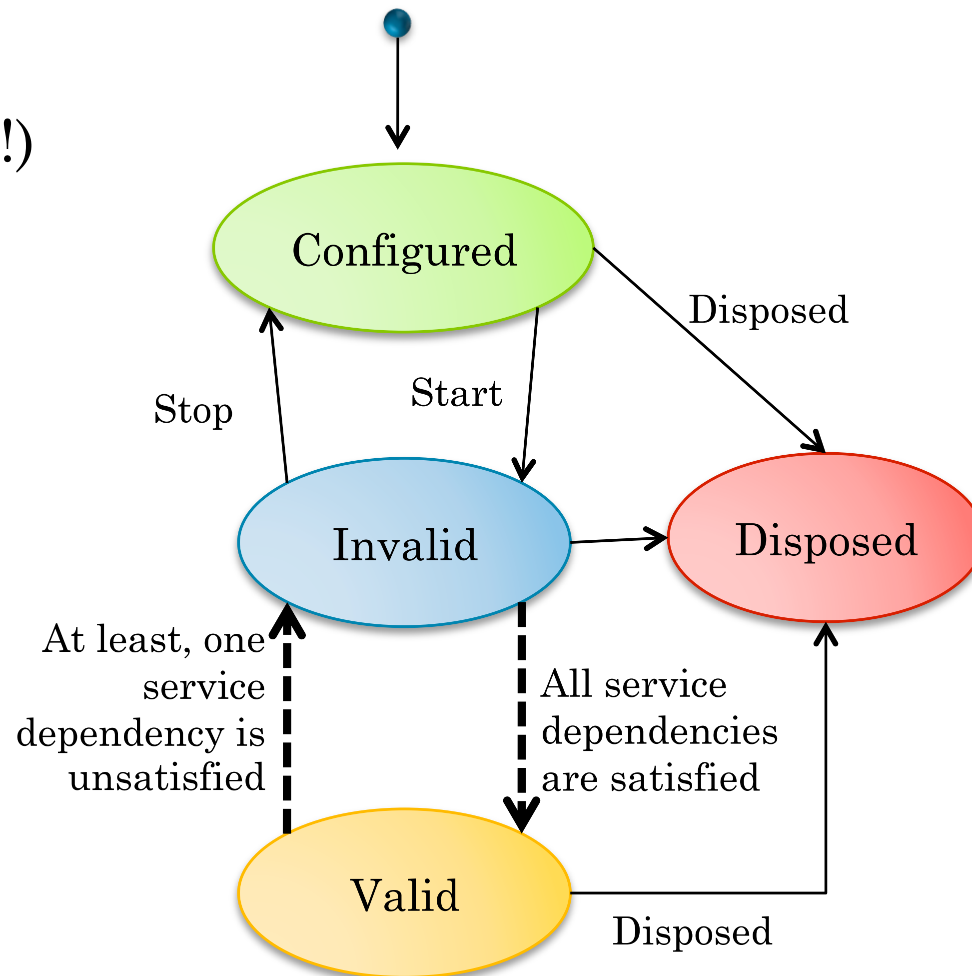
iPOJO, Our Proposal

- 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



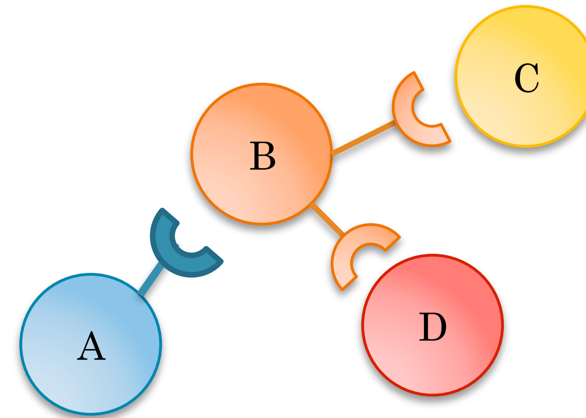
Component Types & Instances

- Component Types
  - Implementations (code!)
  - Describe provided and required services
  - Supports updates
- Instances
  - "*Living*" entities
  - Requires and Provides services
  - *Introspectable*



Service Specification

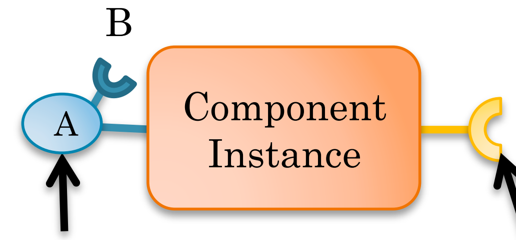
- A service is described with
  - Java interface
  - Properties (open set)
  - State
  - Service dependencies



- Designed to support structural composition
  - Applications are designed using composable services specification

A Rich and Flexible dependency Model

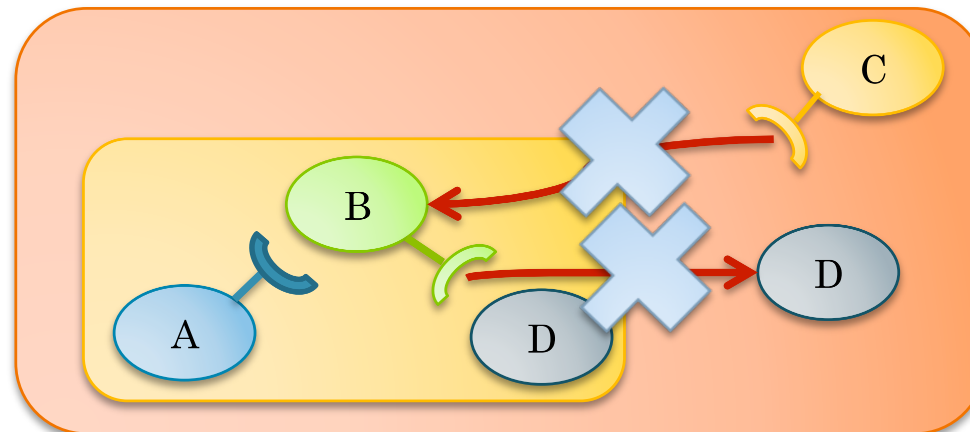
- Two levels of dependencies
  - Service-level
  - Implementation-level



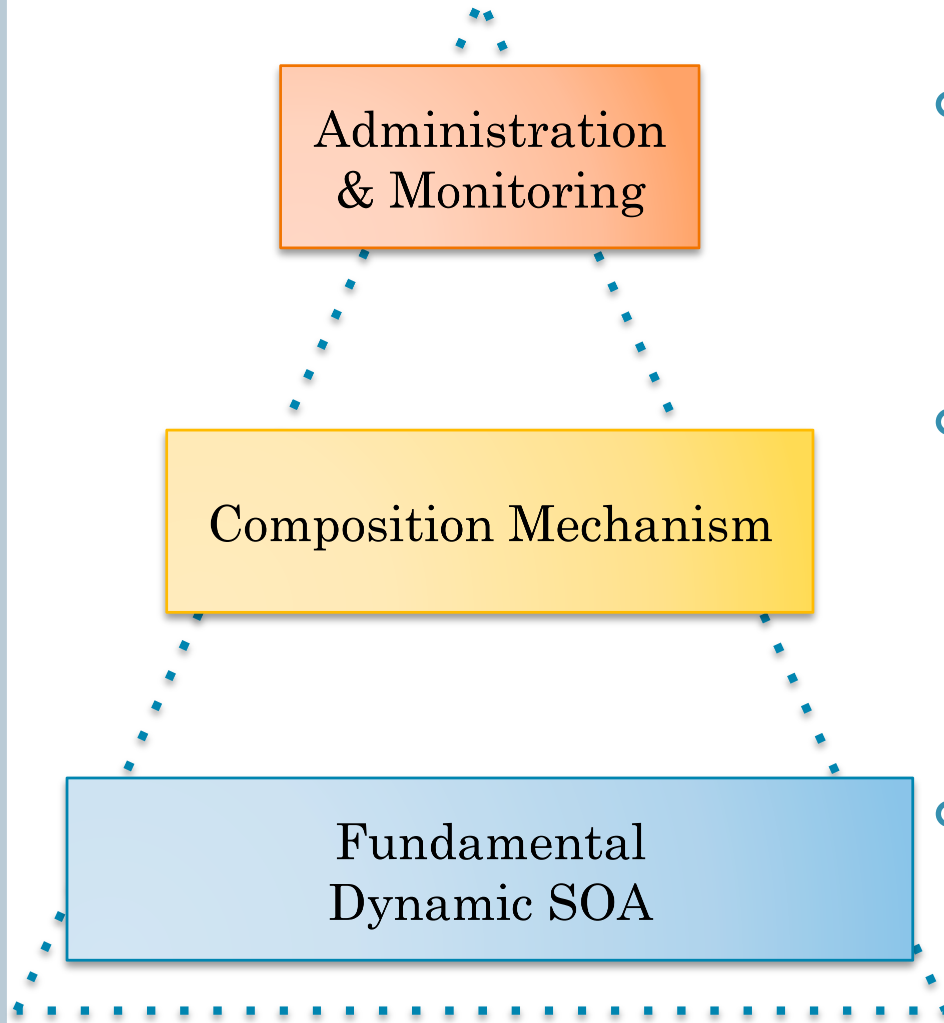
- A dependency targets a service specification
  - scalar or aggregate
  - optional or mandatory
  - can be filtered and/or sorted
  - binding policies
    - Dynamic, Static, Dynamic-Priority
- Properties
  - Reconfigurable, *Introspectable*

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

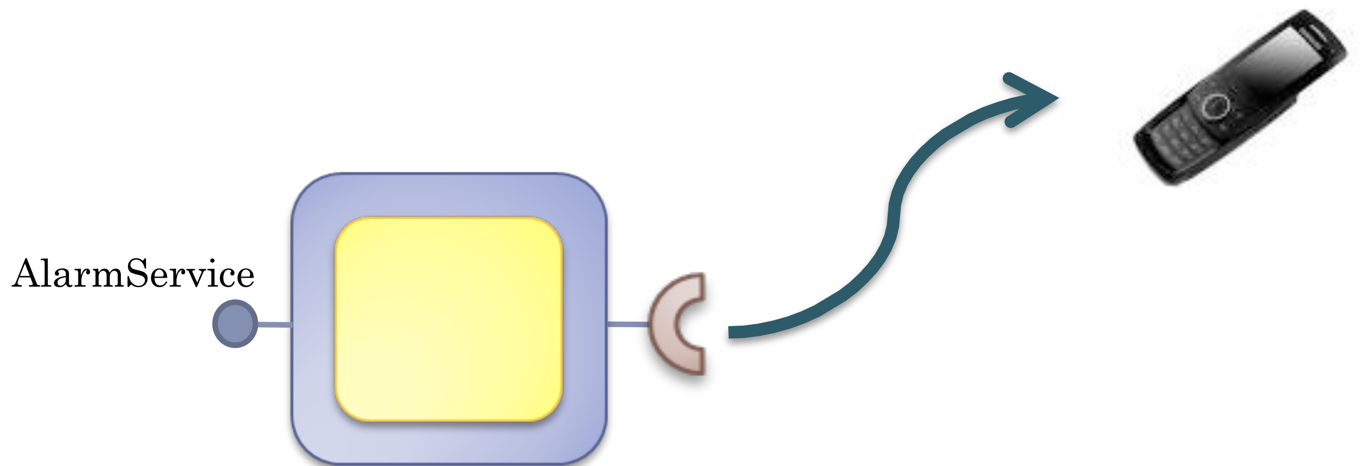
ATOMIC COMPONENT

- 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

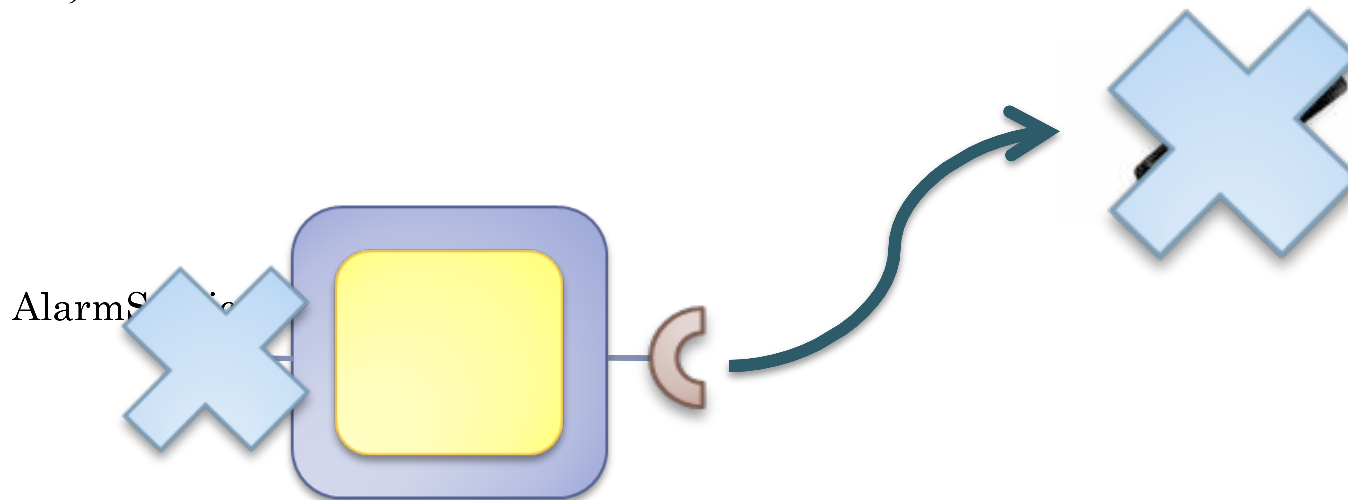
Example Of Atomic Component Description

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



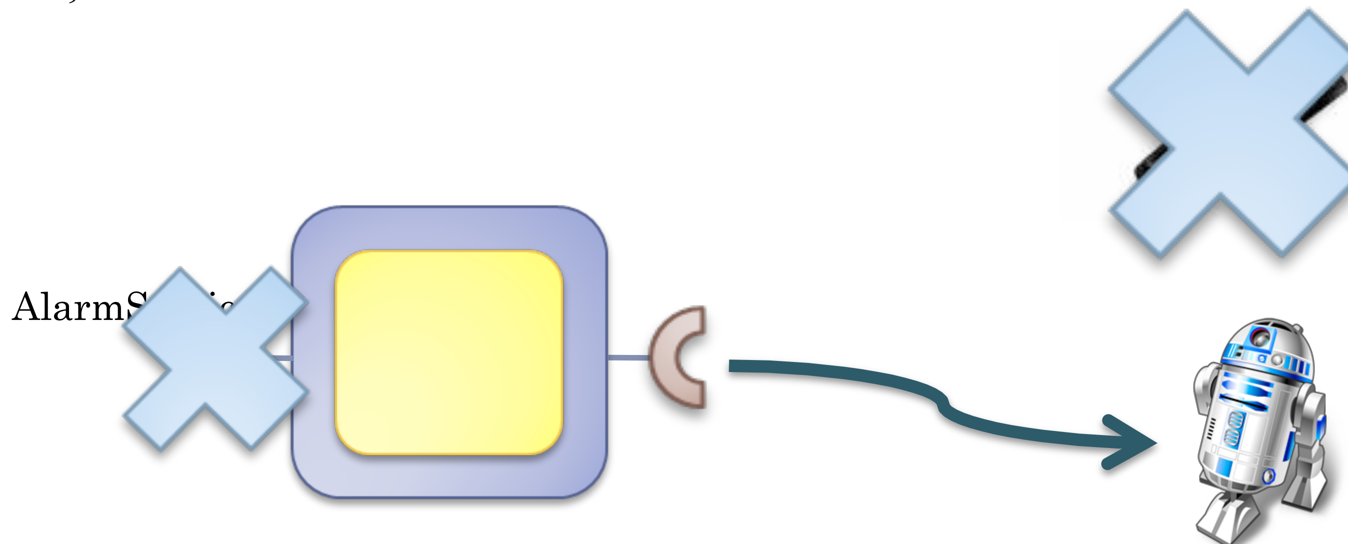
EXAMPLE OF ATOMIC COMPONENT
SERVICE DEPARTURE MANAGEMENT

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



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));
    }
}
```

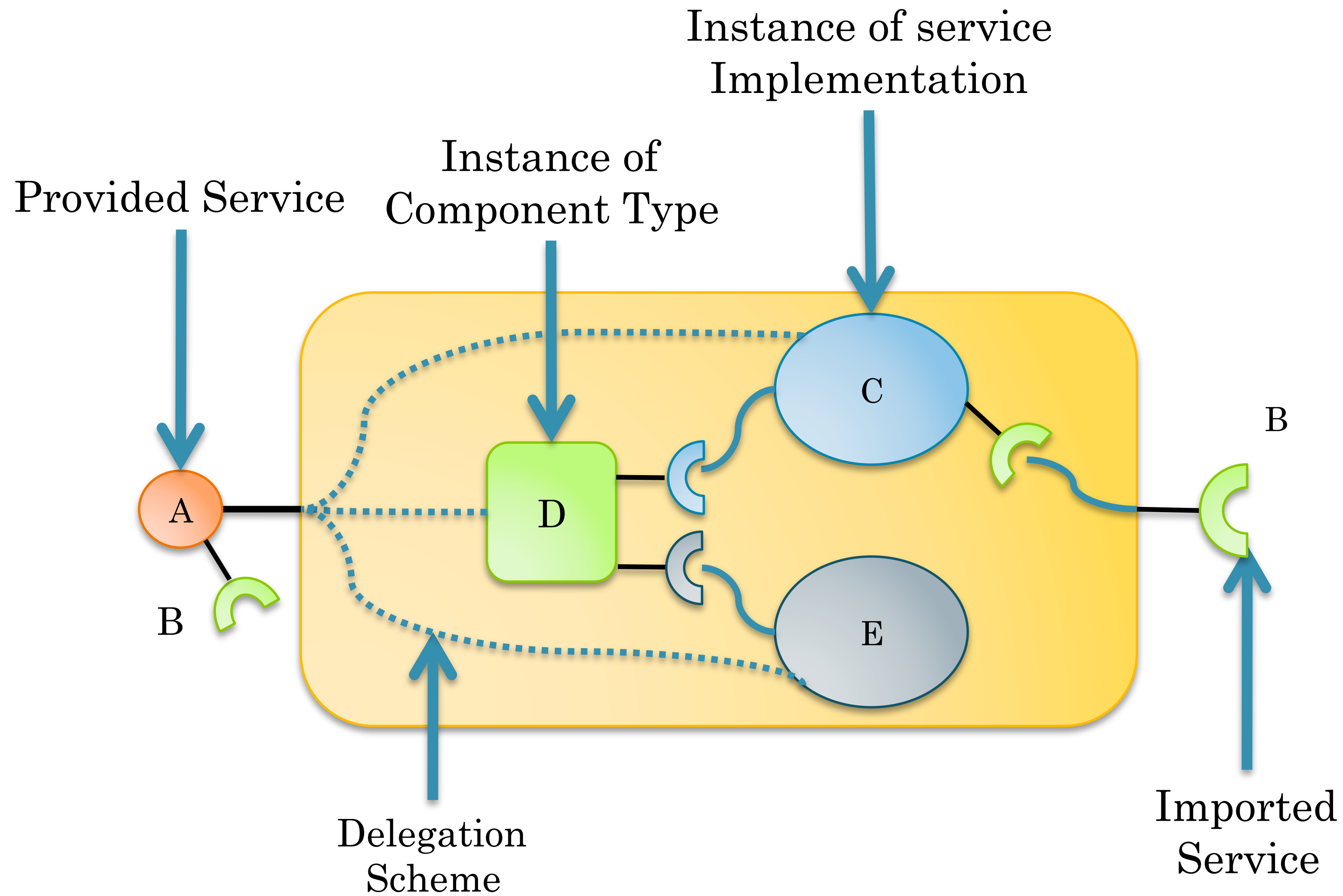


Composite Component

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

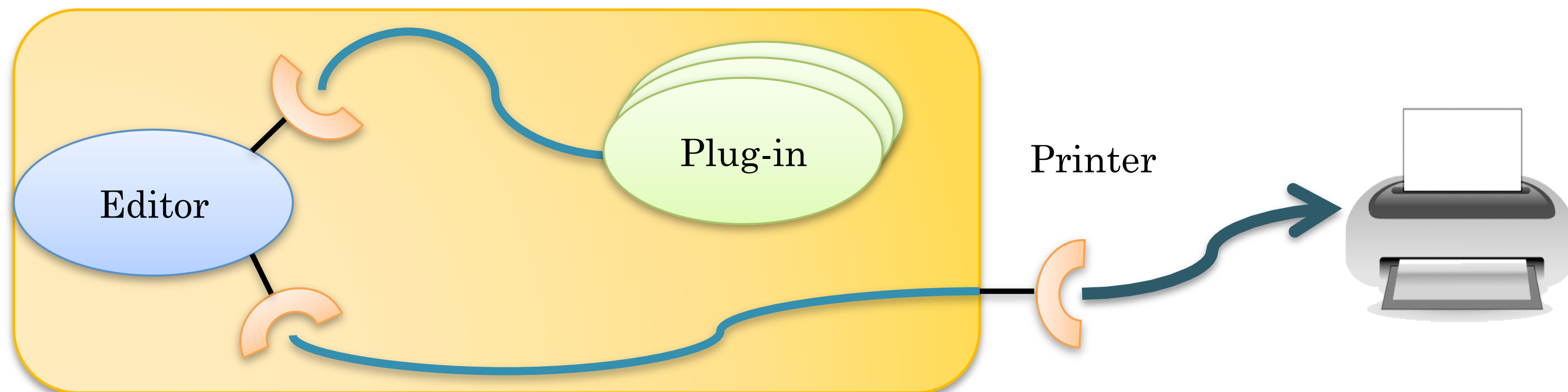
- Characteristics
  - Application concept and vertical composition
  - Implementation evolution and substitution
  - Context-awareness

Composite Component Description



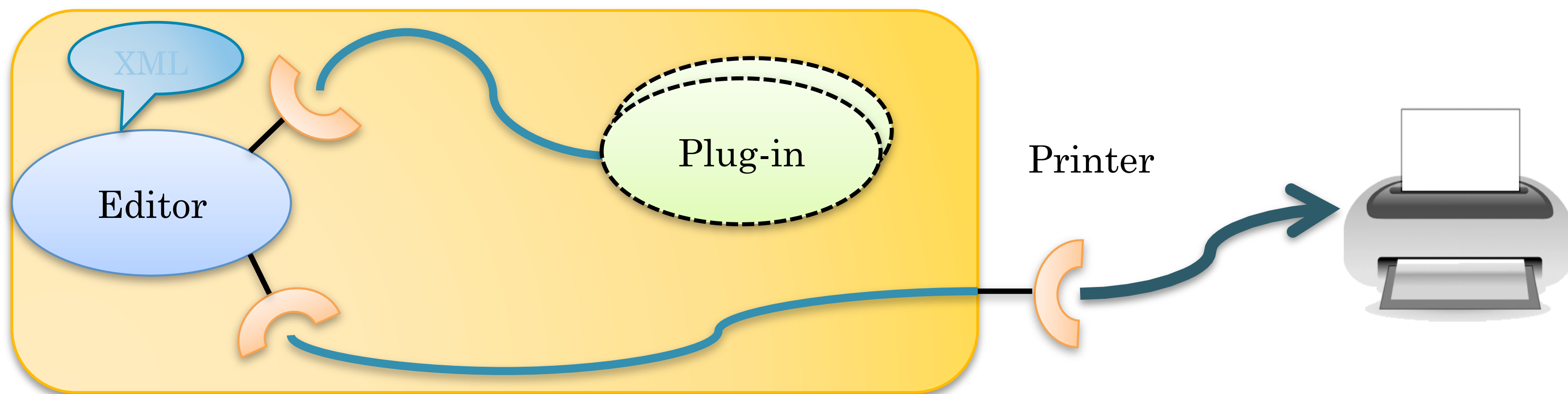
Composite Component Example

```
<composite name="Editor1">
<subservice action="instantiate"
      specification="…Editor"/>
<subservice action="instantiate"
      specification="… Plugin"     aggregate="true" />
<subservice action="import"
      specification="…Printer"   optional="true"/>
</composite>
```



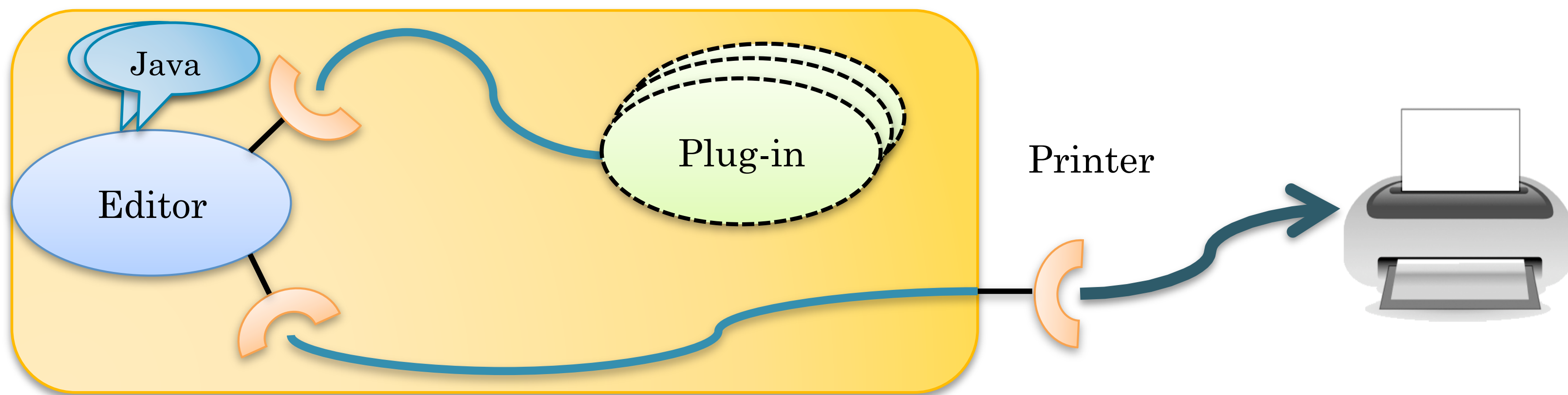
Composite Component
Context-Awareness Example

```
<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>
```



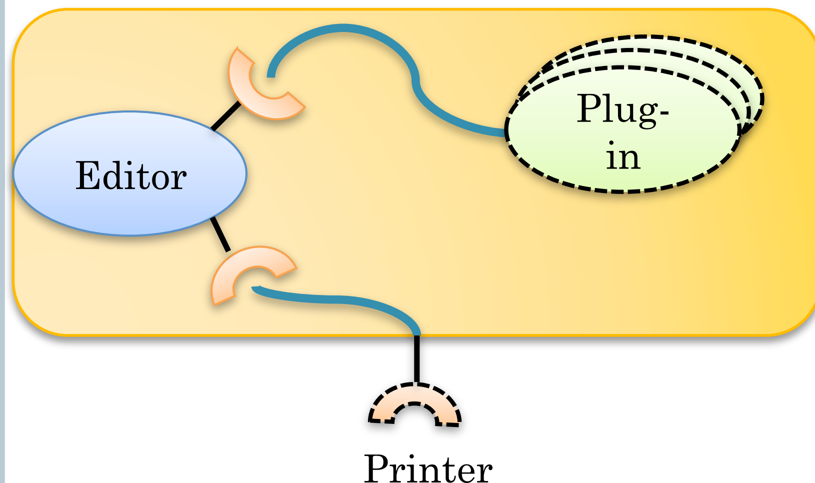
Composite Component
Context-Awareness Example

```
<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

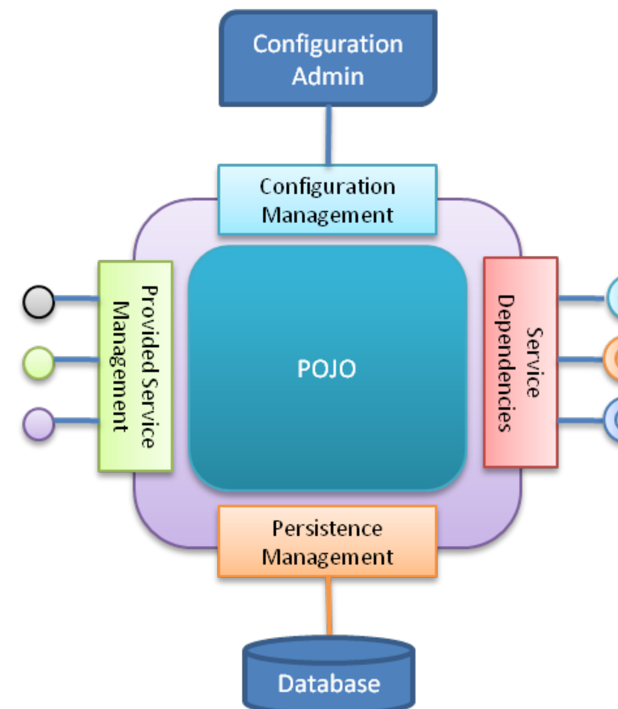



```
<composite name="Editor3">
<subservice action="instantiate"
      specification="… Plugin"  aggregate="true"
      filter="(type=${my.type}) "
      context-source= " local:editor" />
<subservice action="instantiate"
      specification="…Editor"/>
<subservice action="import"
      specification="…Printer"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



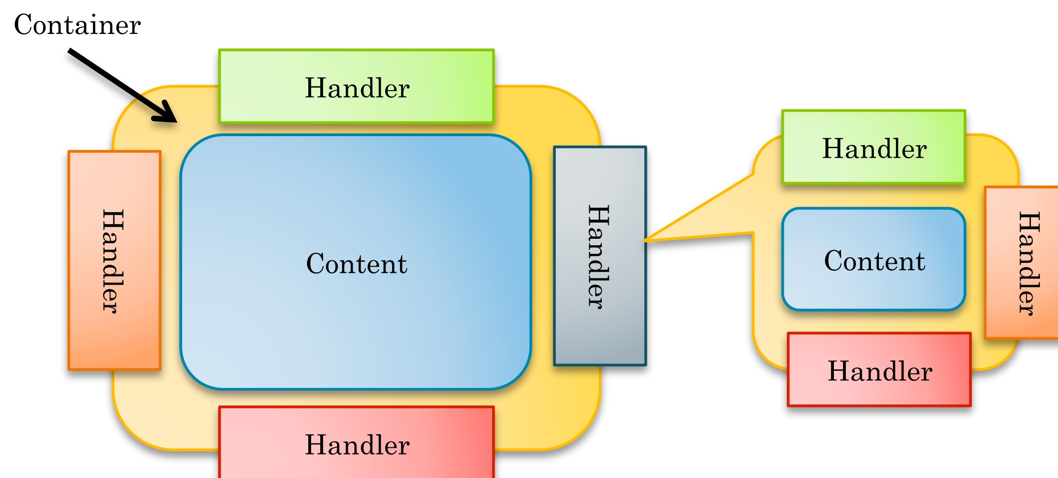
Synthesis

- 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

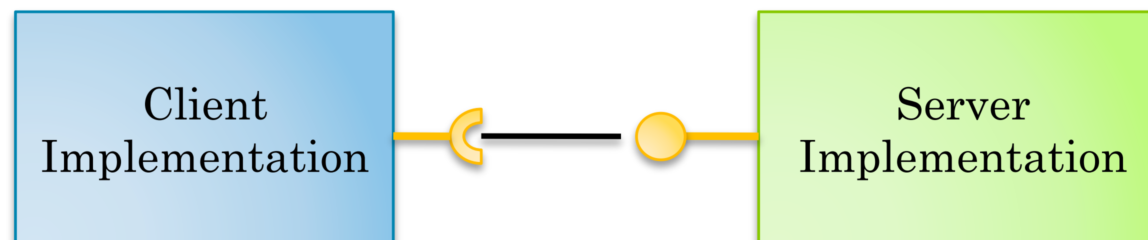
Implementation

- 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

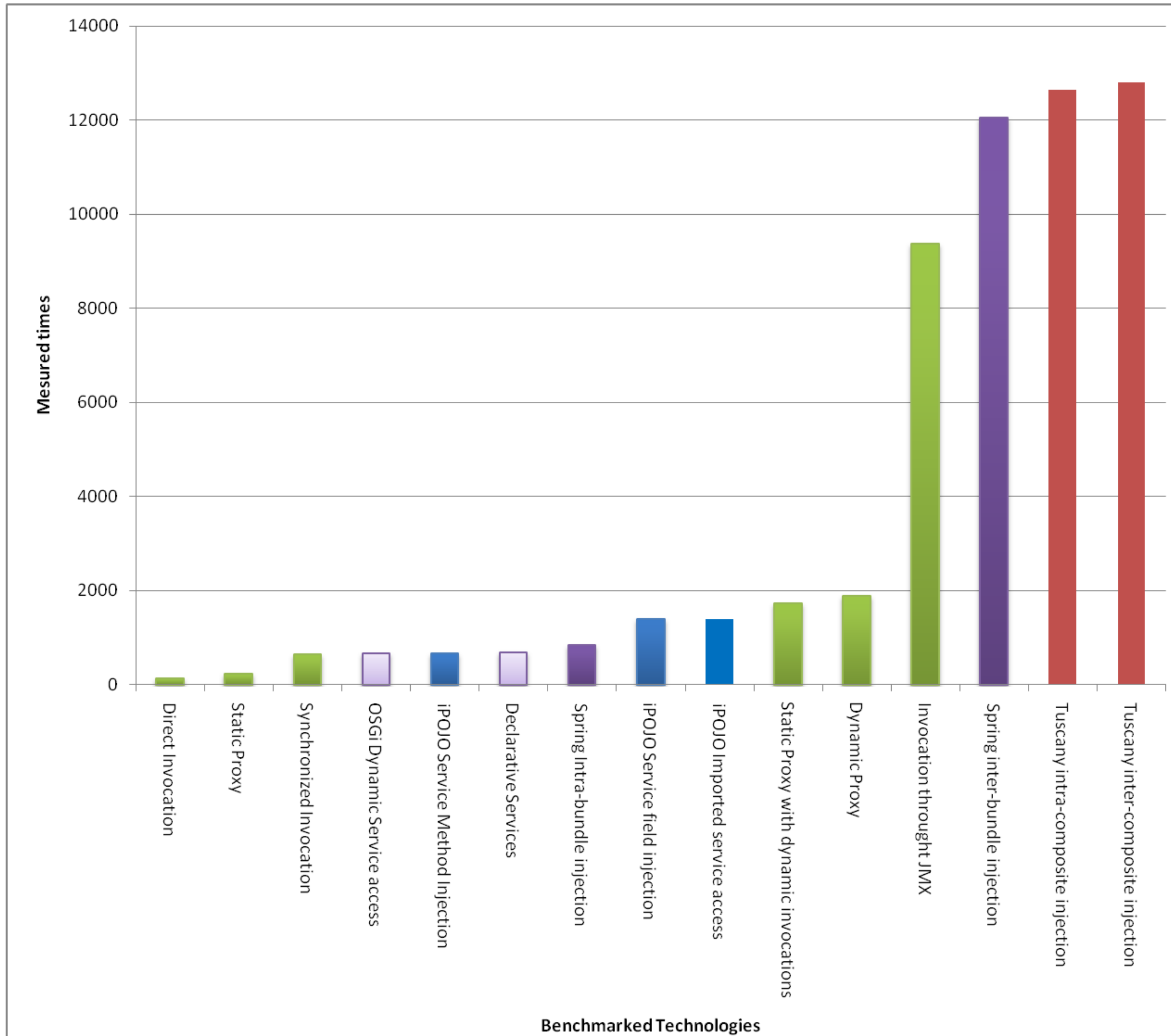


VALIDATION
BENCHMARK

- 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



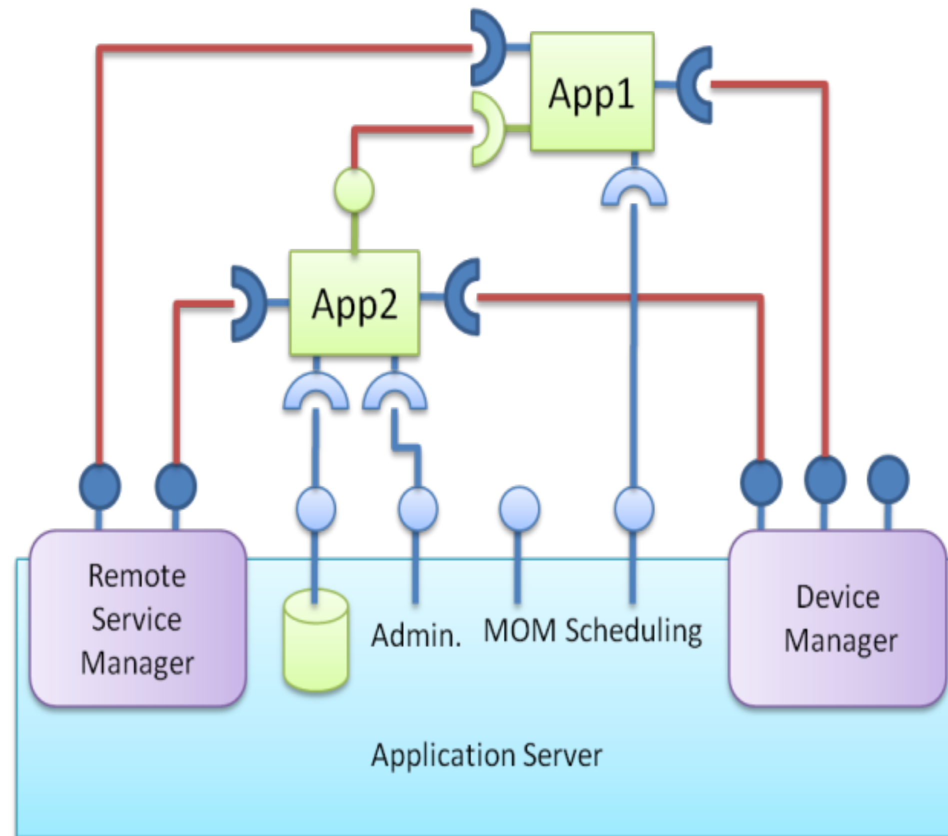
Injection Benchmark Results



RESIDENTIAL GATEWAYS EXAMPLE

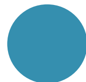
- OSGi/iPOJO framework is used to develop residential gateways.

- Requirements:
  - Dynamism management
  - Extensibility
  - Composition and Isolation

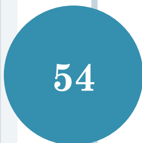


Java EE Server Example

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



Conclusion & Perspectives



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)

http://www.apache.org/

PERSPECTIVES

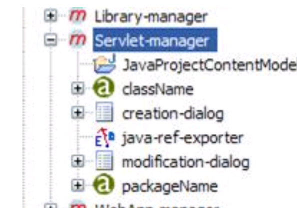
- Apply iPOJO principles on different technologies
  - Principles can also be used on the top of other technologies than OSGi™
  - However, rare are the frameworks providing the required underlying functionalities
- Deployment support
  - How to ease the deployment of dynamic applications?
- Context-Aware and Autonomic Applications
  - iPOJO can be used to execute context-aware and autonomic applications
  - What are the missing features?

PERSPECTIVES

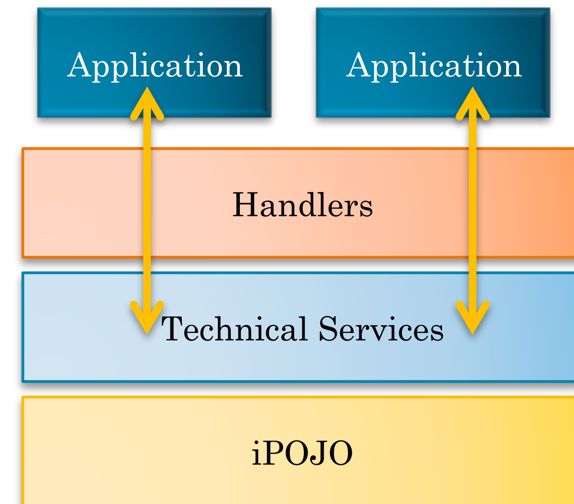
- Domain-driven application servers
  - How to provide an ADL, an IDE and an execution framework for a specific domain
  - iPOJO extensibility mechanisms can be applied to solve such problems.
  - Ongoing …

Specific Composition Language

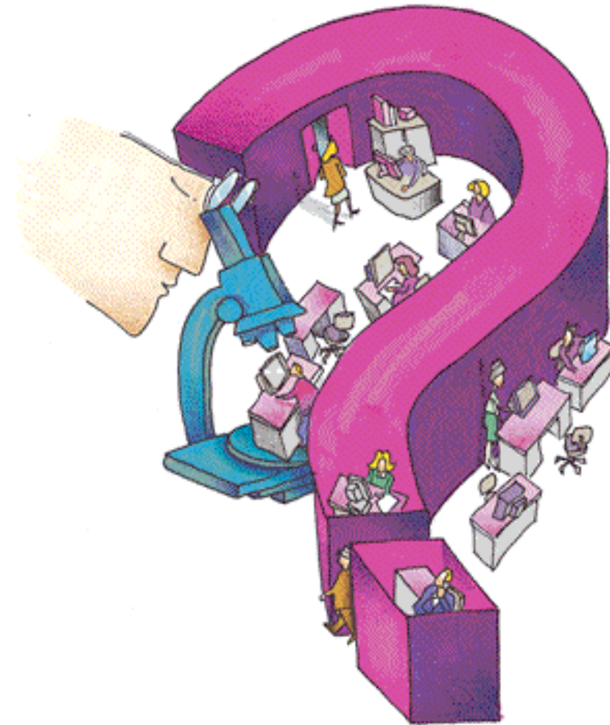
Specialized IDE



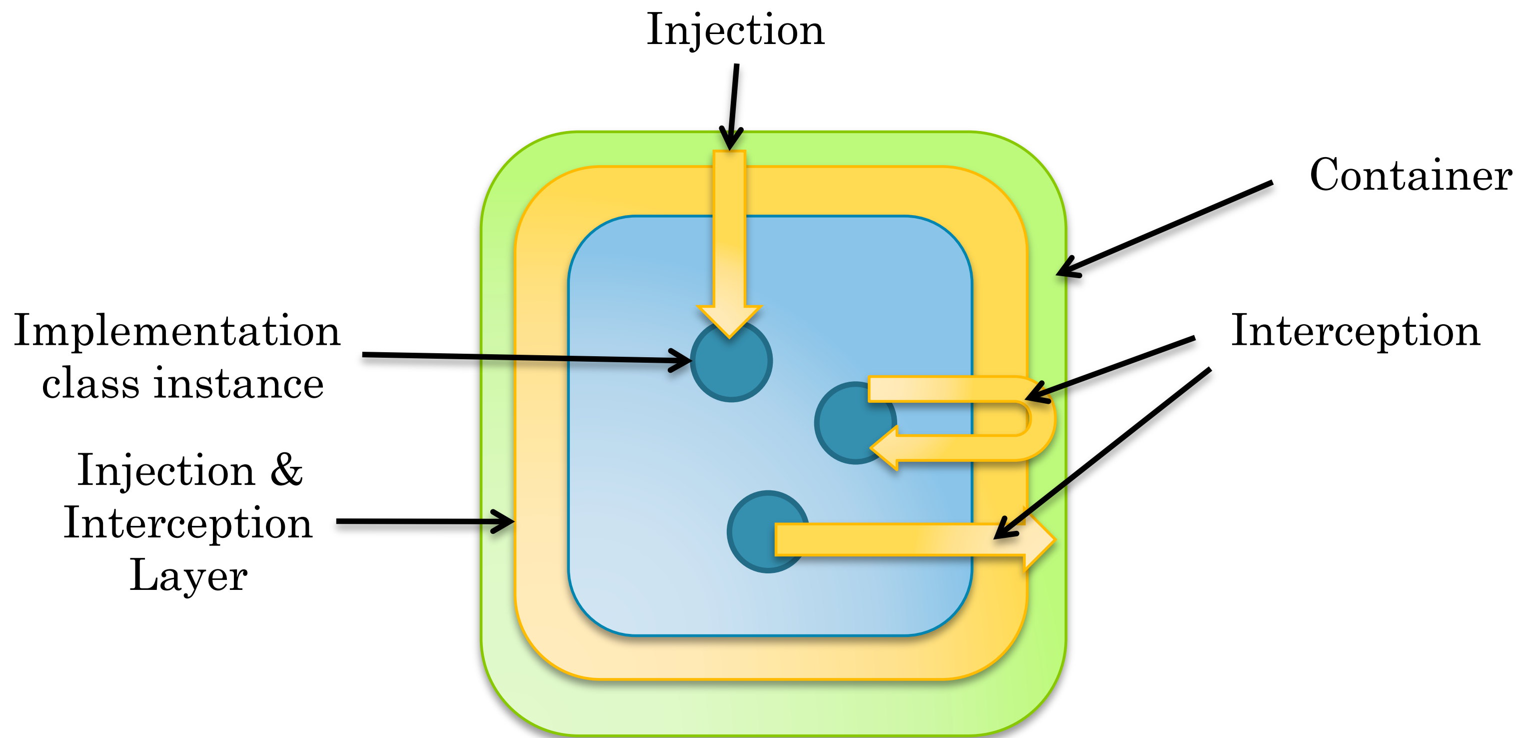
Execution Environment



Q & A



Appendix A
Interception & Injection Framework



APPENDIX A
INTERCEPTION & INJECTION FRAMEWORK

Appendix B
Lines Of Code

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