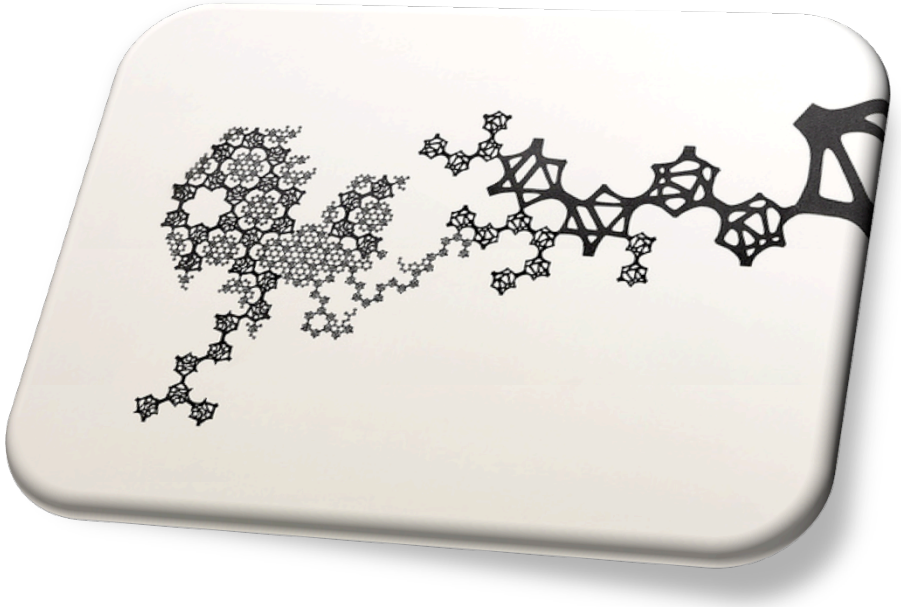
akquinet



iPOJO

A Tale about Simplicity

akquinet

What about me ?

- Solution Architect in the Modular and Mobile CC
- Apache Software Foundation
  - PMC Apache Felix, Apache Ace
  - Apache Felix iPOJO project leader
- OW2
  - Chameleon project leader

The Apache Software Foundation
http://www.apache.org/

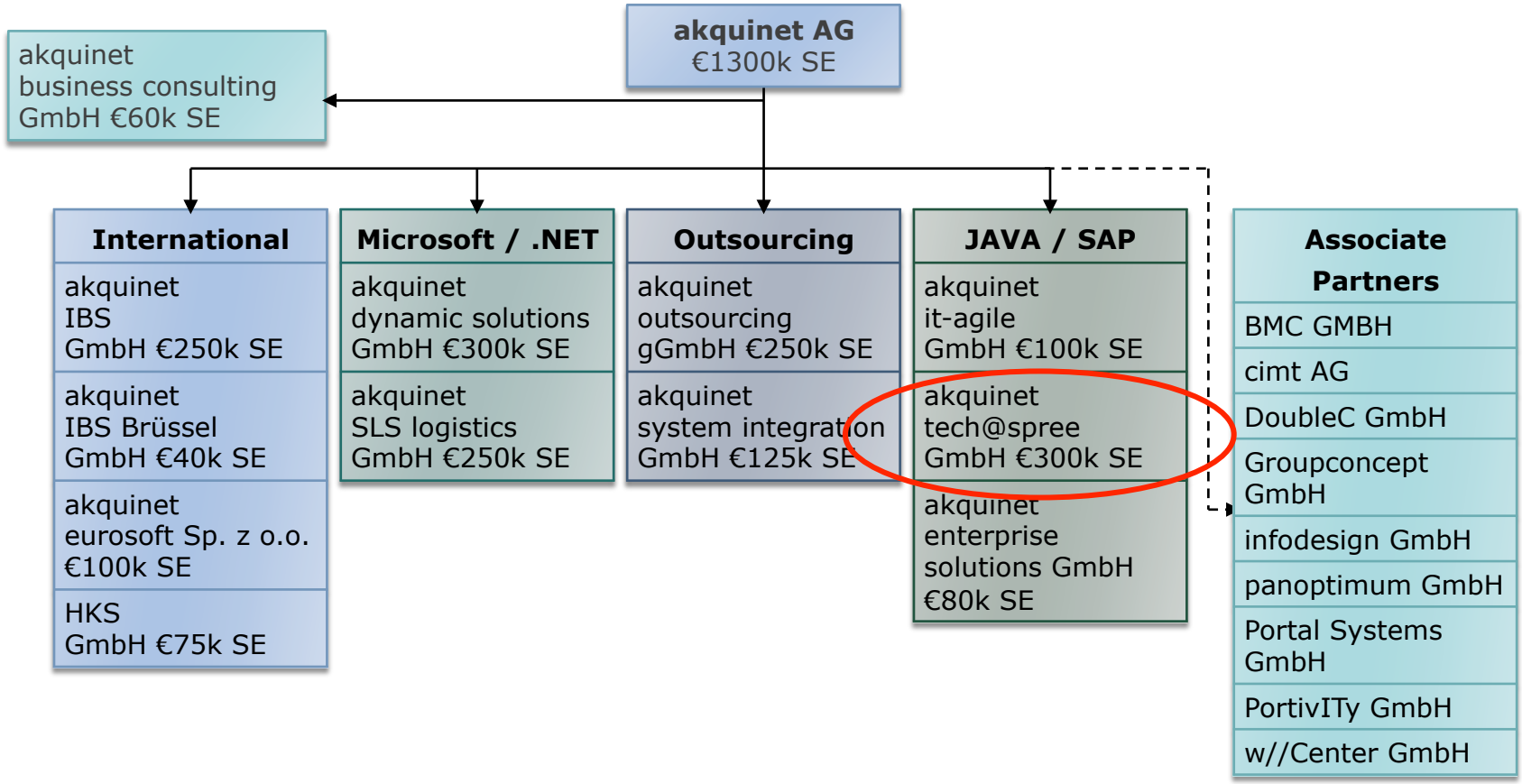
OW2 Consortium

APACHE felix

iPOJO

chameleon

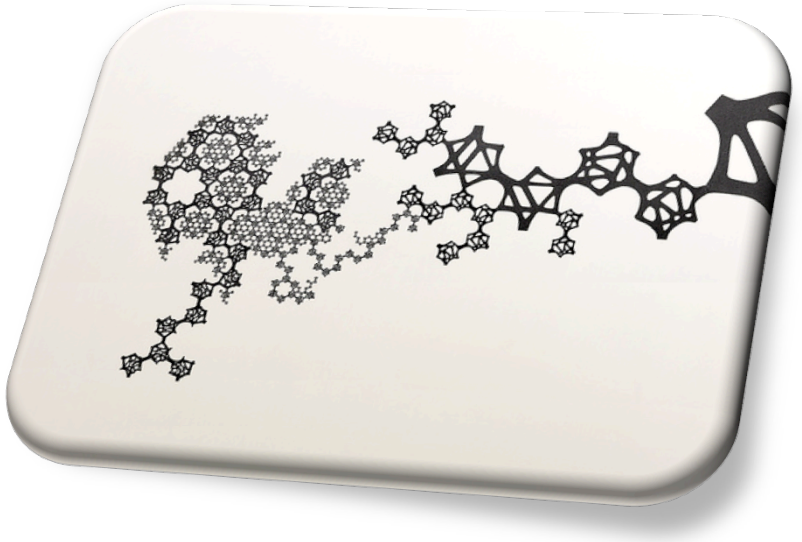
akquinet


akquinet AG
€1300k SE
akquinet
business consulting
GmbH €60k SE
International
akquinet
IBS
GmbH €250k SE
akquinet
IBS Brüssel
GmbH €40k SE
akquinet
eurosoft Sp. z o.o.
€100k SE
HKS
GmbH €75k SE
Microsoft / .NET
akquinet
dynamic solutions
GmbH €300k SE
akquinet
SLS logistics
GmbH €250k SE
Outsourcing
akquinet
outsourcing
gGmbH €250k SE
akquinet
system integration
GmbH €125k SE
JAVA / SAP
akquinet
it-agile
GmbH €100k SE
akquinet
tech@spree
GmbH €300k SE
akquinet
enterprise
solutions GmbH
€80k SE
Associate Partners
BMC GMBH
cimt AG
DoubleC GmbH
Groupconcept GmbH
infodesign GmbH
panoptimum GmbH
Portal Systems GmbH
PortivITy GmbH
w//Center GmbH

akquinet

Modular and Mobile Solutions

- Competence Center focusing on
  - Modular Systems
    - Modularization expertise
    - OSGi-based
    - Sophisticated, Large scale, Distributed systems
  - Mobile Solutions
    - *In the large*
      - Mobile devices, Interactions middleware, Server-side …
      - M2M, B2B
- Open Technologies
  - OSGi (Apache Felix, Apace Ace, OW2 Chameleon, Apache Sling…)
  - Android
  - Java EE (JBOSS, OW2 JOnAS)



Modularity

The Graal Quest ?

akquinet

Time Travel

A well-defined segmentation of the project effort ensures system modularity. Each task forms a separate, distinct program **module**.

At implementation time each module and its **inputs** and **outputs** are well-defined.

At checkout time the integrity of the module is tested **independently**.

Finally, the system is maintained in **modular fashion**.

R. Gauthier and S. Pont, Designing Systems Programs, 1970.

akquinet

Time Travel

The major advancement in the area of modular programming has been the development of coding techniques and assemblers which

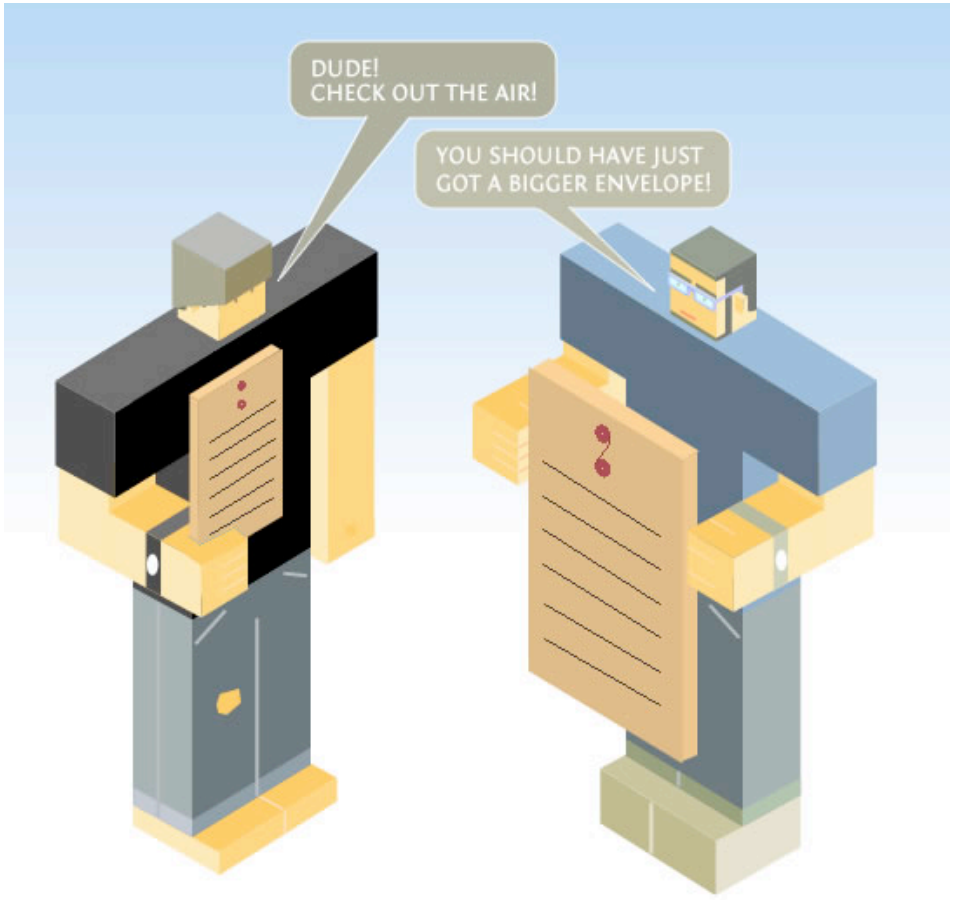
(l) allow one module to be written with **little knowledge** of the code in another module, and

(2) allow modules to be **reassembled** and **replaced** without reassembly of the **whole system**.

D. Parnas, On the Criteria to Be Used in Decomposing Systems into Modules, 1972

My software is bigger than yours !

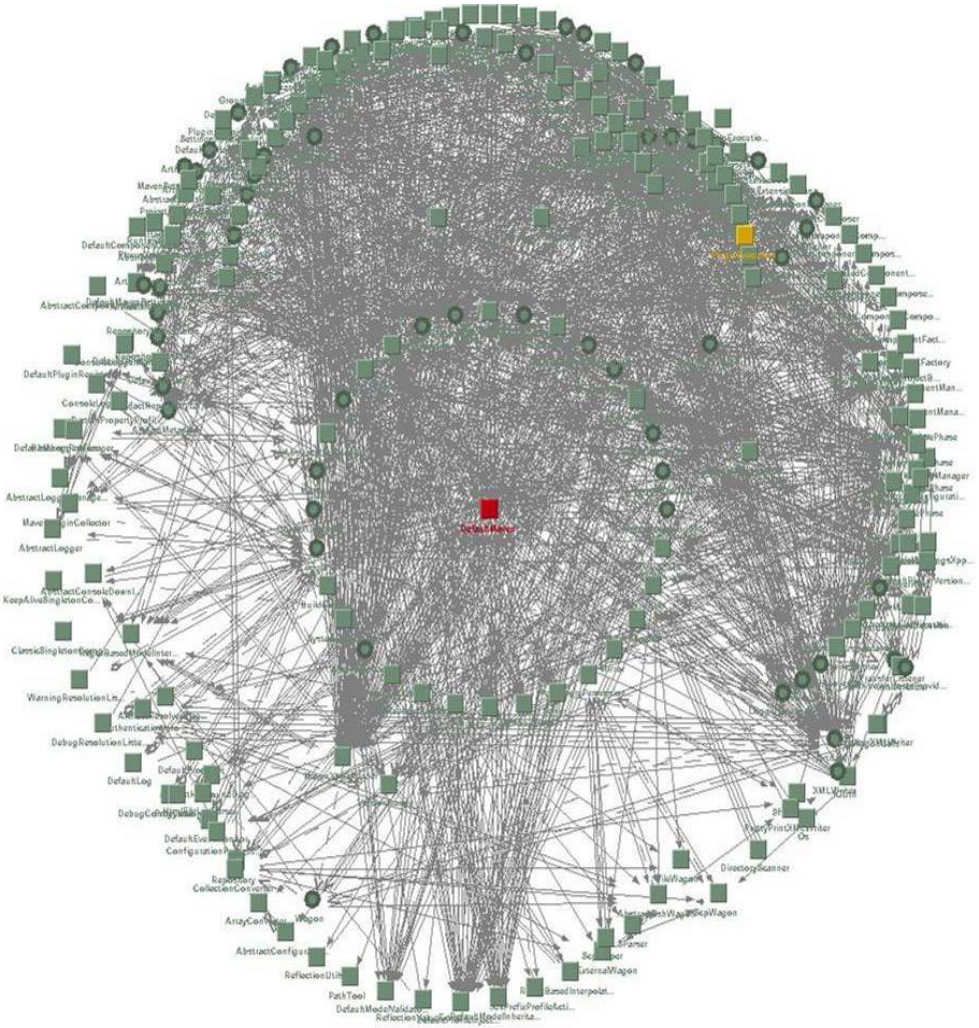
- A DVD player can contain 1 Million lines of code (space shuttle <0.5 million lines)
- A BMW car can contain up to 50 networked computerized devices
- Eclipse contains 3.5 million lines of code
- @ 10 lines a day …
- Libraries are a necessity, but …



akquinet

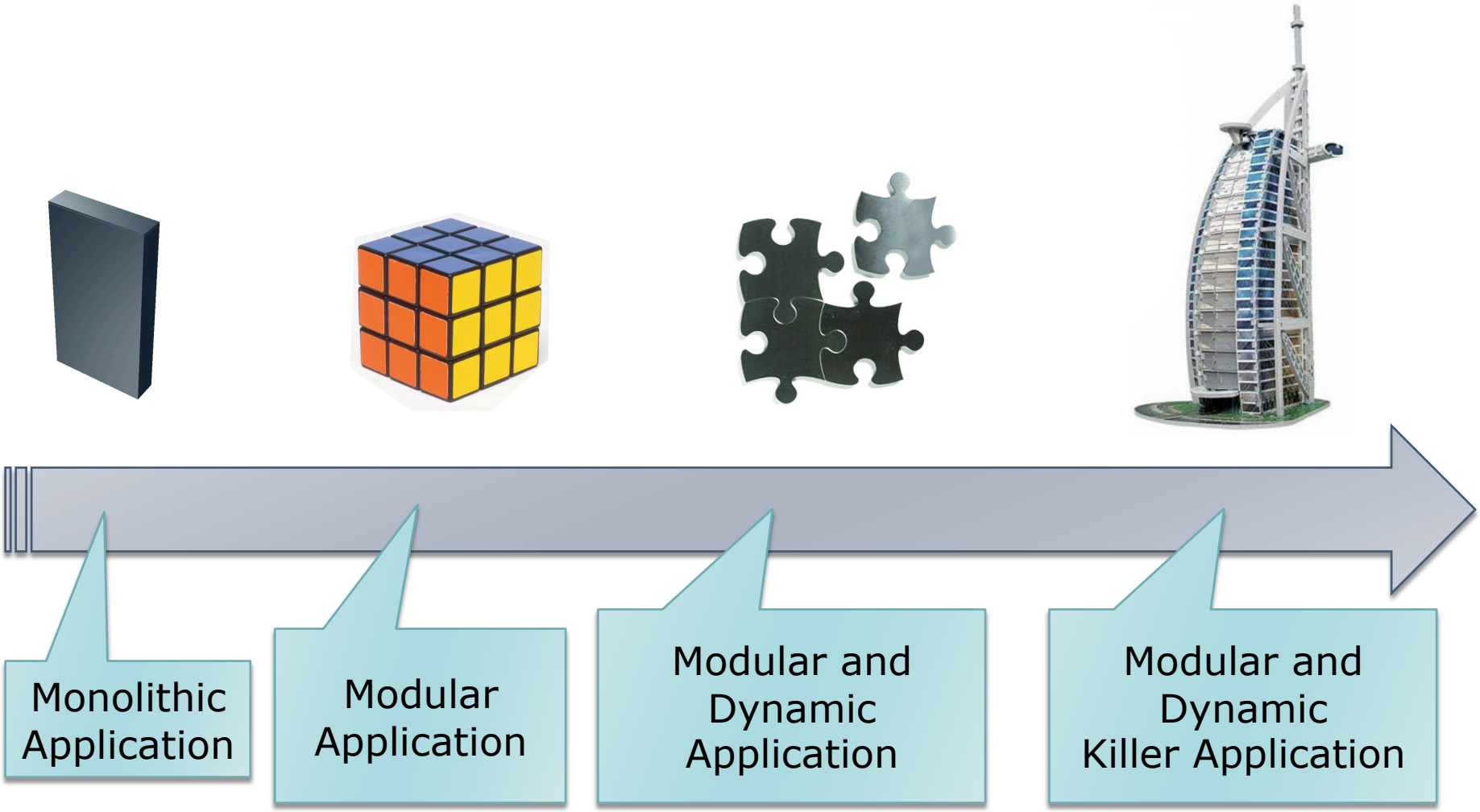
Limits of OOP

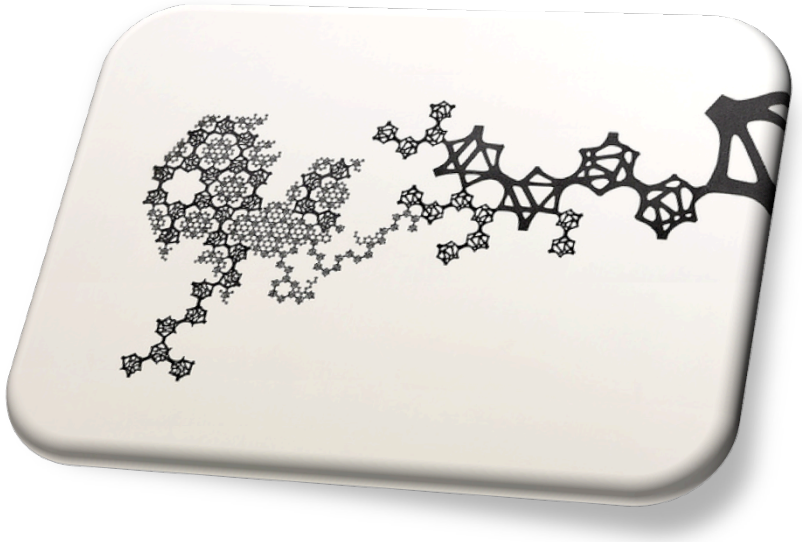
- Coupling severely limits reusability
  - Using a generic object, can drag in a large number of other objects
- Creates overly large systems after a certain complexity is reached
- Flexibility must be built in by the programmer
  - Plugin architectures
  - Factories, Dependency Injection



akquinet

Modularize !





OSGi

The Graal ?

akquinet

Why OSGi ?

- Need simpler ways to construct software systems
  - OSGi is about software construction: building systems out of smaller components …
  - OSGi is about components that work together …
  - OSGi is about managing components …
  - OSGi is about "Universal Middleware"

akquinet

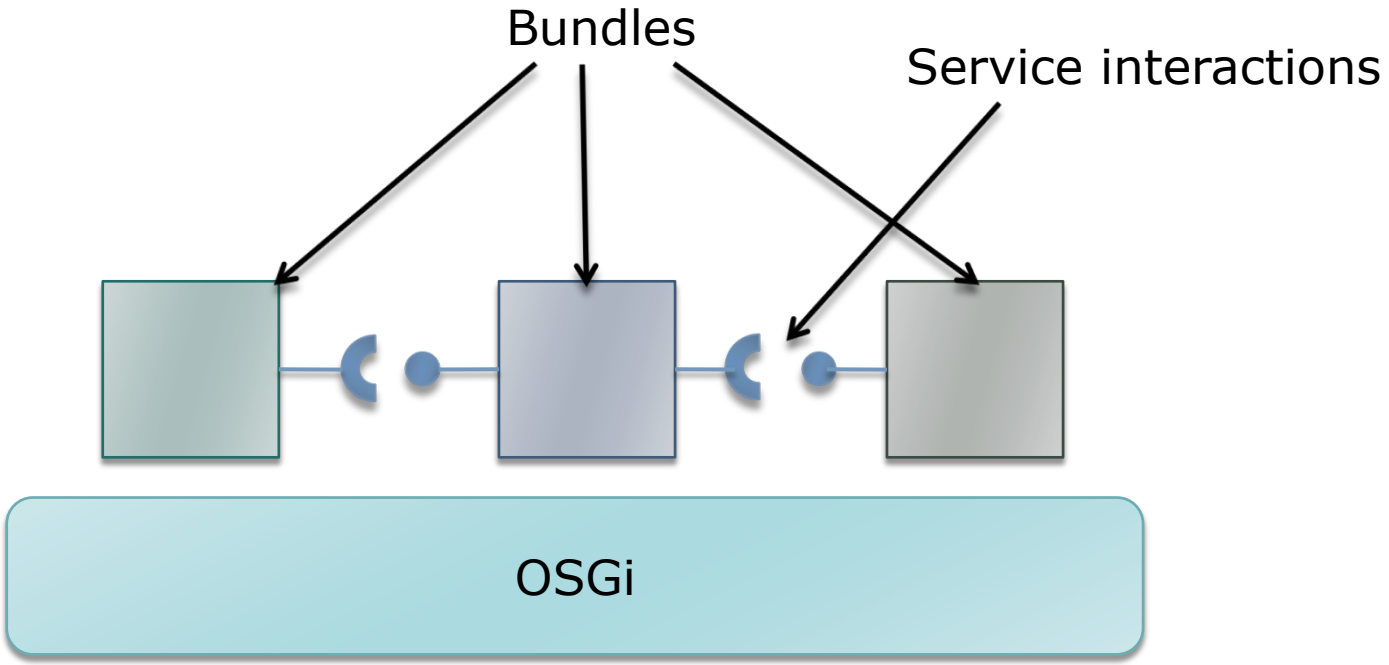
OSGi to build modular systems

- OSGi
  - Module layer on the top of Java
  - Define
    - A module layer
      - Bundle
      - Relation between bundles (import / export packages)
    - A lifecycle layer
      - Define bundle lifecycle
      - Continuous deployment
    - A service layer
      - Dynamic, loose-coupled interactions between bundles
      - Promotes substitution

akquinet

OSGi to build modular systems

- Modular system with OSGi
  - Define the bundles and the interactions
  - Deploy
  - Update



akquinet

Does it work ?

- For (very) small system :
  - Yes
- For complex system
  - It's a Nightmare
  - *Classloaders on steroids*
    - Classloader hell
    - Systems were not design to follow these rules
      - How to integrate legacy ?
  - OSGi API is complex
  - **Dynamism is challenging**

akquinet

Does it work ?

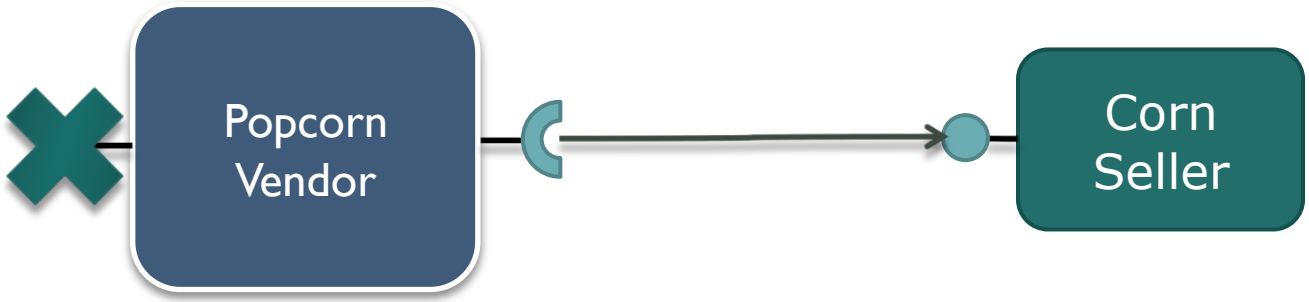
- With the OSGi momentum, more and more systems are OSGi powered
  - Linkedin
    - completely hides OSGi, is it really modular ?
  - Module Fusion
    - OSGi enterprise distribution
    - Goes a little further, but a lot of issues are still there
      - Try to change your JPA entity classes…
  - Eclipse
    - Why I have to reboot ?
    - Why I can’t install the plugin P version X ?

OSGi is not enough – recipe errata

- Before building sophisticated modular applications on the top of OSGi, we need:
  - An OSGi framework (R4.2 compliant)
  - A set of technical / common services
    - Compatible
    - Flexible
  - **A component model flexible and extensible**
    - Dynamism pain reliever
    - Integrating technical services in the application
  - Tools
    - Deployment
    - Management
  - Design Guideline

Practical Explaination

- OSGi™ Snack Bar
  - Pop Corn Vendor



akquinet

Popcorn Vendor service listener

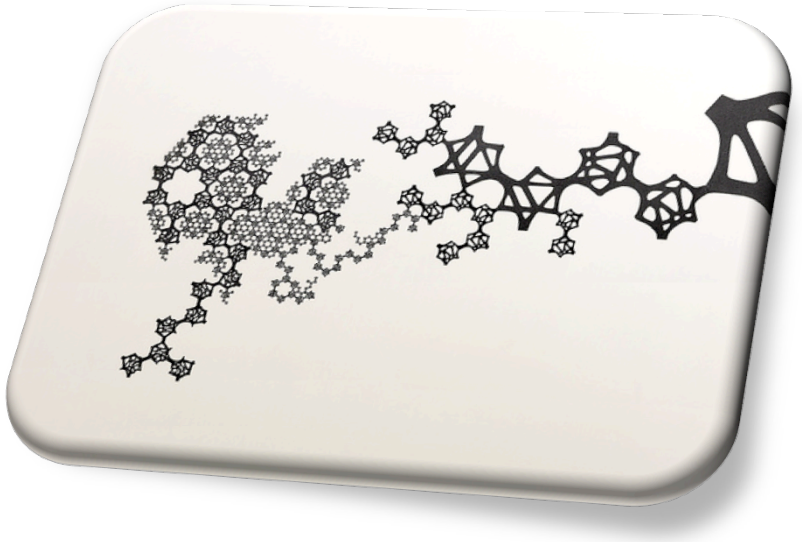
```
public synchronized void serviceChanged(ServiceEvent event) {
        if (event.getType() == ServiceEvent.REGISTERED) { // If a  seller was registered, see if we need one. If
                                                          // so, get a reference to it.
                if (m_ref == null) { // Get a reference to the service object.
                        m_ref = event.getServiceReference();
                        m_seller = (Vendor) m_context.getService(m_ref);
                        m_sr = m_context.register(Vendor.class.getName(), this, null);
                }
        }
        // If a seller was unregistered, see if it was the one we were using. If so, unget the service and try to query
        // to get another one.
        else if (event.getType() == ServiceEvent.UNREGISTERING) {
                if (event.getServiceReference() == m_ref) { // Unget service object and null references.
                   m_context.ungetService(m_ref);
                   m_ref = null;
                   m_seller = null;
                   // Query to see if we can get another service.
                   ServiceReference[] refs = m_context.getServiceReferences(Vendor.class.getName(), "(type=corn)");
                   if (refs != null) {
                        m_ref = refs[0]; // Get a reference to the first service object.
                        m_seller = (Vendor) m_context.getService(m_ref);
                   } else { // No more provider.
                        m_sr.unregister();
                    }
        }
}
```

akquinet

Do you see a problem ?

- A little complex, isn't it ?
  - Listeners are difficult to implement correctly
  - Do not forget to release used services
  - Mix dynamism management and business logic

- Question ?
  - How to ease OSGi™ application development
    - Development
    - Design



(Service) Component Model

The Graal Trail ?

akquinet

Component Models

- Objectives:
  - Beyong object programming …
  - Separation of concerns / Abstraction
    - Persistance, Security, Bindings
  - Avoid monolithic applications
    - Application compozed by components
    - An assembler select components and bind them
  - Hierarchical composition
    - Component can contain other components …

Component Models : Principles

Objects

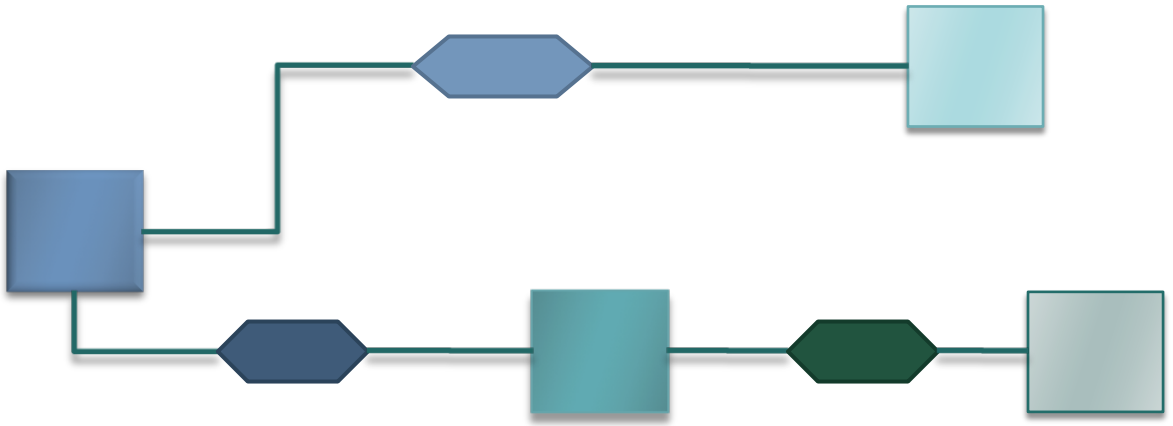
- Class : Object type
  - Code unit
  - Encapsulation
- Instance : Object

Components

- Component type
  - Unit of code or composition
  - Non-functional concerns description
  - Provided & Required services
- Component Instance
  - One content
  - One container

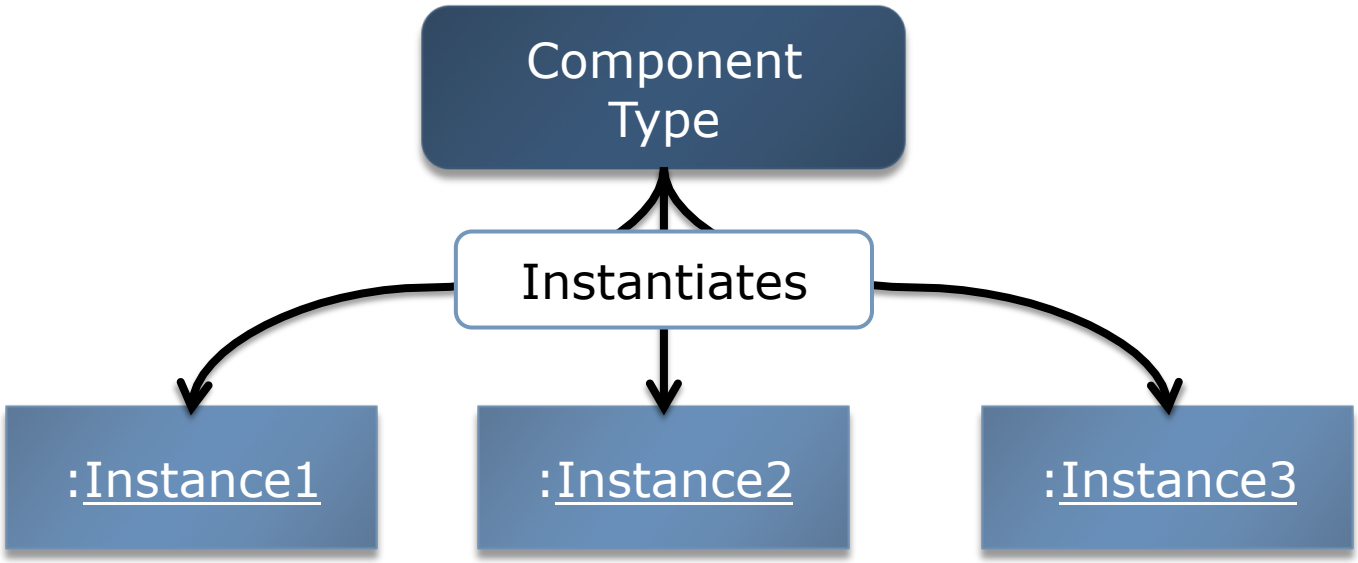
Component Models : Principles

- Architecture Description Language
  - How to describe an application
    - Selection of component & binding (configuration)
  - Static checking
    - Matching between provided and required interfaces

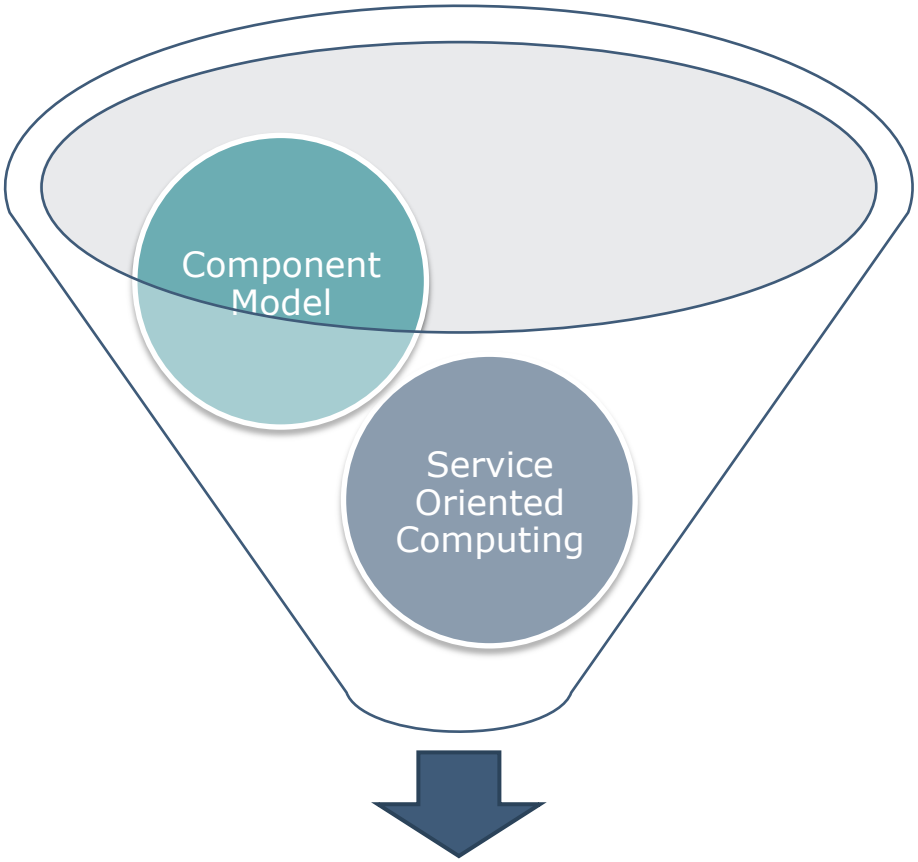


Component Models : Factories

- Factories
  - There is no « new » on component
    - Instantiation of the container and of the content
  - Principles of factories
    - Allow component instances creation



Service Component Model (SOCM)



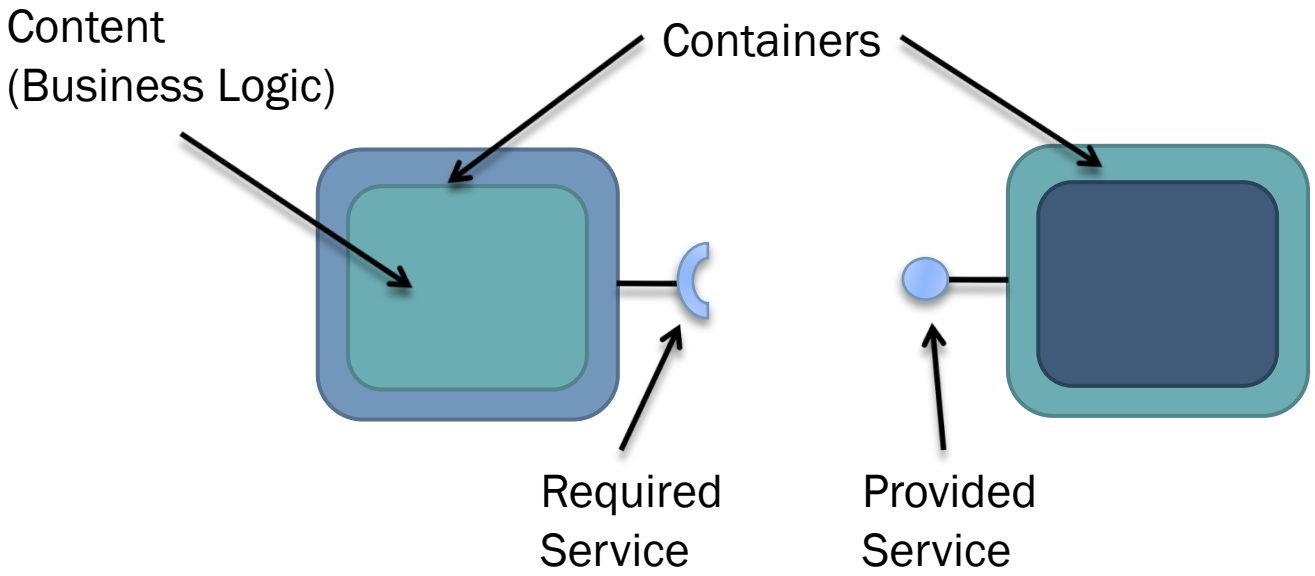
akquinet

SOCM : Principles

- An instance
  - Provides services
  - Requires and uses services
- Service dependencies are resolved at runtime by following service-oriented computing paradigm
- Composition in term of service specifications

SOCM

- Basically, SOCM containers manage
  - Required services (discovery, selection, binding, tracking)
  - Provided services (publication, withdrawal, service object creation …)



akquinet

Existing SOCM for OSGi™

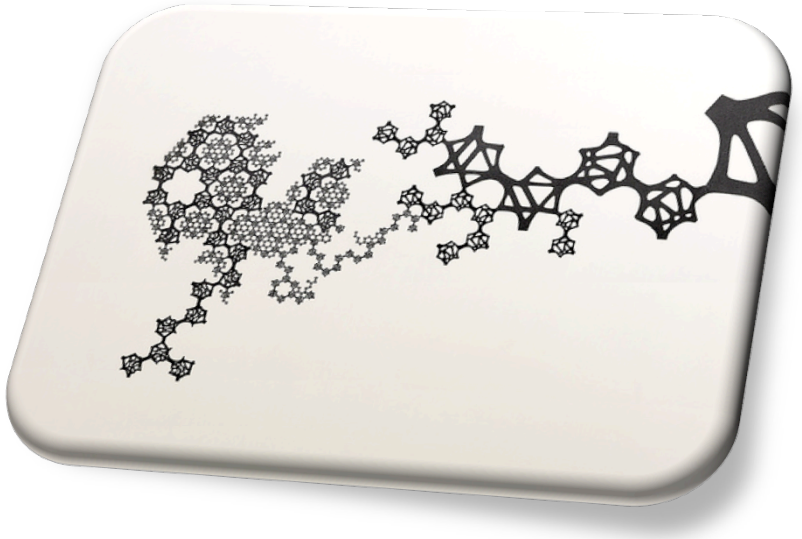
- Service Binder / Declarative Service :
  - OSGi™ R4 (Compendium)
  - Define de component model managing service dependency and providing
    - Required services are injected via “bind” and “unbind” methods
    - Instance lifecycle is managed by the container
- Dependency Manager
  - API to develop component
  - Managing service providing and service requirements
  - Allow to add dynamically new dependencies

akquinet

Existing SOCM for OSGi™

- Blueprint
  - Spring framework on the top of OSGi™
  - Injection of OSGi™ services
  - Manage others non-functional concerns (persistence, transaction, configuration …) [not-standardized]

- iPOJO
  - Extensible component model
  - Supports annotations, XML and API descriptions
  - Real separation type / instance
  - Manages service dependencies and service providing ….
  - Allow to add others non-functional properties (JMX, persistence, security …)
  - Provide an ADL to build service based applications

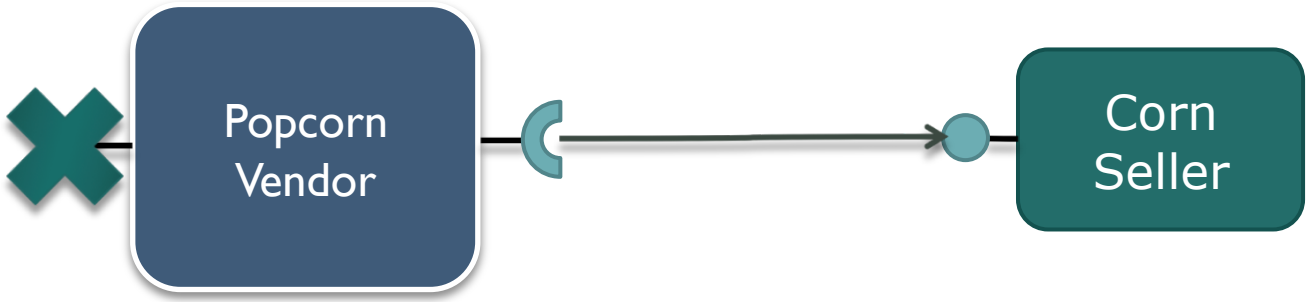


iPOJO

The fastest path ?

Snark Bar Reloaded

- OSGi™ Snack Bar
  - Pop Corn Vendor



Implementation using iPOJO

```
@Component(name="popcorn-provider")
@Provides
public class PopcornVendor implements Vendor {
    @Requires
    private Reseller m_vendor;

    public  String getName() { return "Popcorn from Paris"; }
    public String buy() {
            m_vendor.buy();
            return "popcorn";
    }
}
```

```
<ipojo>
        <instance component="popcorn-provider"/>
</ipojo>
```

akquinet

Implementation using next iPOJO version

```
@Component(name="popcorn-provider")
@Instantiates
@Provides
public class PopcornVendor implements Vendor {
    @Requires
   private Reseller m_vendor;

   public  String getName() { return "Popcorn from Paris"; }
   public String buy() {
          m_vendor.buy();
          return "popcorn";
   }
}
```

akquinet

iPOJO

- Apache Felix sub-project
- iPOJO provides
  - A very simple development model (POJO)
  - The automation of service based interactions
    - Publication, Service Properties, Service discovery, Service tracking, Service binding
  - An extensible infrastructure
    - To manage other non functional features

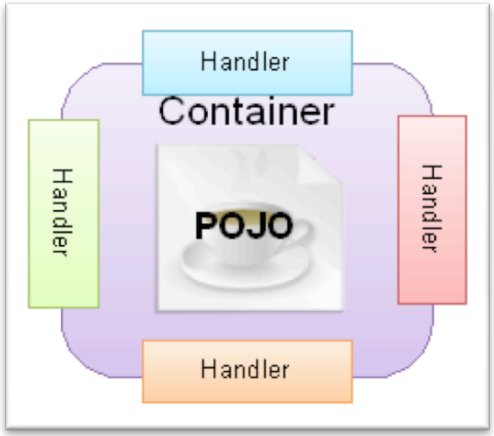
akquinet

iPOJO overview

- Differentiation between
  - Component types (<component>)
    - Describe non-functional requirement
  - Instances (<instance>)
    - You can create several instances from the same component types
    - Will really provide services, required services…
- Injection mechanism
  - Container can interact with the POJO by
    - Invoking methods on the POJO (Container => POJO)
    - Injecting value inside fields (Container => POJO)
    - Be notified of POJO change on field (POJO => Container)

Mechanism & Container

- Bytecode weaving
  - Aspect-like mechanisms
- Metatada format agnosticism
  - XML, Annotation, API
- Highly Extensible
  - Handlers



akquinet

Component Type

- @Component
  - Declares a component type
  - Target class definition

```
@Component
public class MyClass {…}
```

- Attributes
  - Name : factory name
  - Immediate : is the instance immediately activated (boolean)
  - Propagation : enables / disables configuration propagation
  - …

akquinet

Service Providing

- @Provides
  - All implemented interfaces will be published
  - Target class definition

```
@Component
@Provides
public class MyClass implements MyService {…}
```

- Attributes
  - Specifications: set the published interfaces (class [])
  - Factory: Service Object creation
    - {SINGLETON|FACTORY|INSTANCE|…}

akquinet

Service Properties

- @ServiceProperty

```
@Component(immediate=true)
@Provides
public class Circle implements SimpleShape {

  @ServiceProperty(name=SimpleShape.NAME_PROPERTY)
  private String m_name = "Circle";

  @ServiceProperty(name=SimpleShape.ICON_PROPERTY)
  private ImageIcon m_icon =
    new ImageIcon(this.getClass().getResource("circle.png"));

  public void draw(Graphics2D g2, Point p) {
    ...
  }
}
```

akquinet

Requiring a service

- iPOJO Service requirement
  - Field injection and Method injection
  - Optional or Mandatory
  - Scalar or Aggregate
  - Filter
  - Binding policy : static, dynamic, dynamic priority

Method Injection of Services

- @Bind and @Unbind

```
@Bind(filter="(name=main)")
protected void bindWindow(Window window) {
  m_log.log(LogService.LOG_INFO, "Bind window");
  window.addWindowListener(this);
}

@Unbind
protected void unbindWindow(Window window) {
  m_log.log(LogService.LOG_INFO, "Unbind window");
  window.removeWindowListener(this);
}
```

akquinet

Method Injection of Services

- Attributes
  - Aggregate, Filter, Optional

```
@Bind(aggregate=true)
public void bindShape(SimpleShape shape, Map attrs) {
  final DefaultShape delegate = new DefaultShape(shape);
  final String name = (String) attrs.get(SimpleShape.NAME_PROPERTY);
  final Icon icon = (Icon) attrs.get(SimpleShape.ICON_PROPERTY);
  m_shapes.put(name, delegate);
  SwingUtils.invokeAndWait(new Runnable() {
    public void run() {
      ...
    }
  });
}
```

akquinet

Field injection of Services

- @Requires
  - Manage synchronization for you

```
@Requires(optional=true)
private LogService m_log;
To access the log service, we just use the field, like this:
@Override
public void windowClosed(WindowEvent evt) {
  try {
    m_log.log(LogService.LOG_INFO, "Window closed");
    m_context.getBundle(0).stop();
  } catch (BundleException e) {
  }
}
```

akquinet

Field injection of Services

- Optional management
  - Nullable
  - Default-Implementation
  - Disabled with nullable=false

```
@Requires(optional=true)
private LogService m_log;
To access the log service, we just use the field, like this:
@Override
public void windowClosed(WindowEvent evt) {
  try {
    m_log.log(LogService.LOG_INFO, "Window closed");
    m_context.getBundle(0).stop();
  } catch (BundleException e) {
  }
}
```

akquinet

Field injection of Services

- Synchronization management

```
@Requires Foo m_foo;
public void statelessAccess() {
m_foo.doA()
m_foo.doB()
m_foo.doC()
}
```

akquinet

Lifecycle

- iPOJO Instance state
  - INVALID | VALID

```
@Validate
protected void activate() {
  SwingUtils.invokeAndWait(new Runnable() {
    public void run() {  setVisible(true); }
  });
}

@Invalidate
protected void deactivate() {
SwingUtils.invokeLater(new Runnable() {
        public void run() { setVisible(false); dispose(); }
        });
}
```

akquinet

Configuration

- Component type declare Property
  - @Property
  - Method: setX(value)
  - Field
- Component Instance receive configurations
  - Instance configuration (XML)
  - 'Factory' configuration (API)
  - Configuration Admin
- @Instantiates does not allow configurations

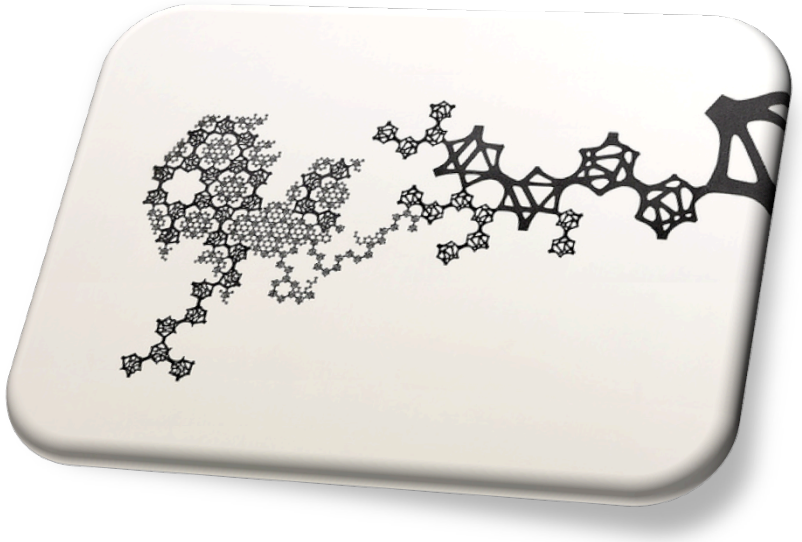
Instance Configuration

- @Component => Component type
- <Instance> => Instance
  - metadata.xml
    - Not necessary in the same bundle
  - Supports configuration

```
<instance component=”foo”/>

<instance component="buns_and_wieners">
    <property name="buns" value="9"/>
    <property name="wieners" value="8"/>
</instance>
```

- Cfg files
  - Java properties file



Conclusion

How are you ?

akquinet

Modularity, OSGi, SOCM

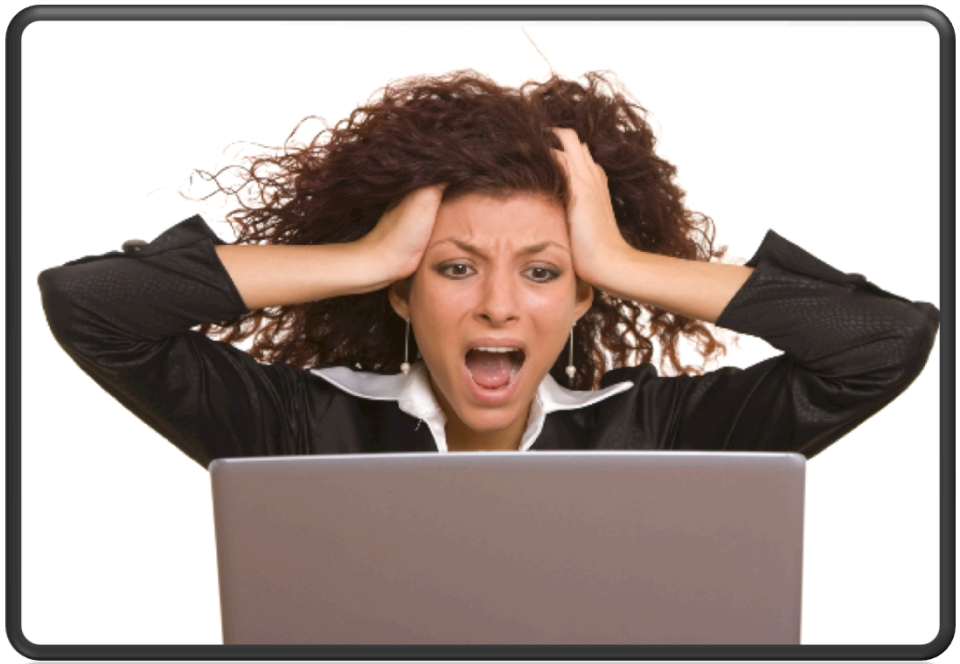
- The objectives
  - Dynamic, modular applications
- The trail
  - OSGi
- The easiest way
  - SOCM

akquinet

iPOJO

- Apache Felix sub-project
  - 5 years of development
  - Used in numerous projects
    - Application server
    - Gateway
    - Mobile phone
- It’s the most advanced component model for OSGi

Questions ?



Karl Pauls
karl.pauls@akquinet.de
Bülowstraße 66, 10783 Berlin
+49 151 226 49 845

Dr. Clement Escoffier
clement.escoffier@akquinet.de
Bülowstraße 66, 10783 Berlin
+49 175 2467717