

Java Services in OSGi - A Practical SOA Example

Jan S. Rellermeyer

Systems Group, Department of Computer Science, ETH Zurich



SOA at your service

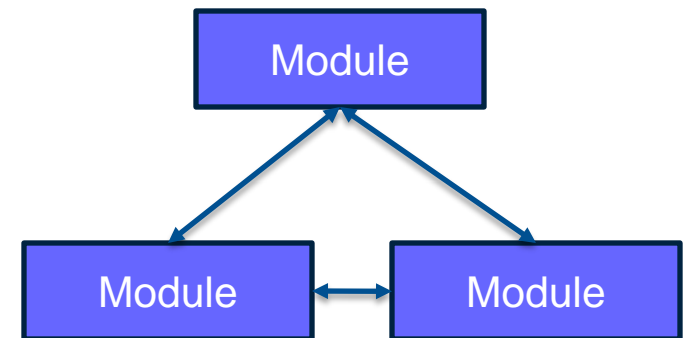
- What's SOA for you?
- Wikipedia has many descriptions and definitions
Service-orientation aims at a loose coupling of services with operating systems, programming languages and other technologies that underlie applications.

(Newcomer, Eric; Lomow, Greg (2005). *Understanding SOA with Web Services*. Addison Wesley)

- Is SOA = interfaces + modularity?

OSGi – Modularity for Java and SOA in a small scale (?)

- Module Management
 - Modules = Reusable parts
 - Manage your Java dependencies
 - Extensible applications
 - Plugins as in Eclipse
- Services
 - Factor out common tasks
 - Loose coupling of Modules



Module Management

- Traditional Java: The mystical class path
- `java -cp commons-X.jar foo.jar bar.jar my.application.MainClass`
 - Which module contains the main class?
 - What are the dependencies between `foo.jar` and `bar.jar`?
 - What happens if `bar.jar` is upgraded to `bar-1.01.jar`?
- The Java model of modularity is weak and fragile!

Module Management in OSGi

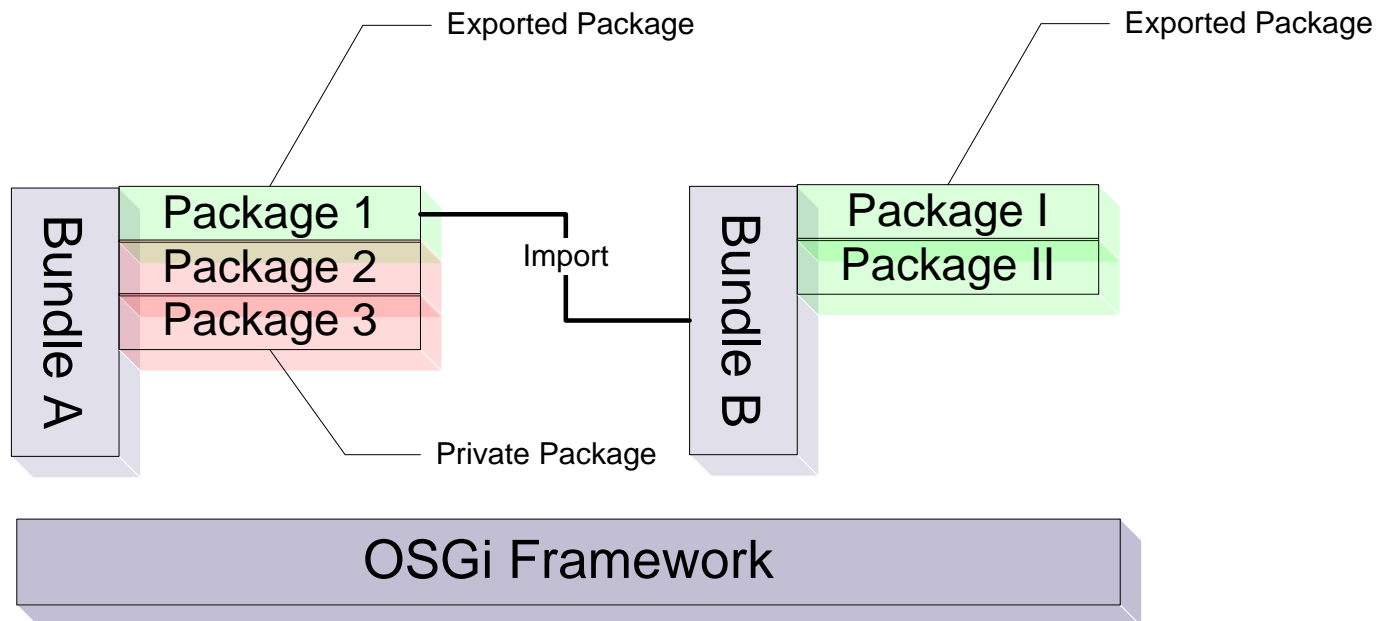
- Modules in OSGi = “Bundles”
- Bundle = JAR file plus additional manifest information
 - Package-Imports/Exports
 - Internal class path
 - Versioning
- Everything is a Bundle.
- Bundles can be active through a **BundleActivator**
 - Name of the Activator class is also in the manifest.

The OSGi Framework

Export-Package: Package 1

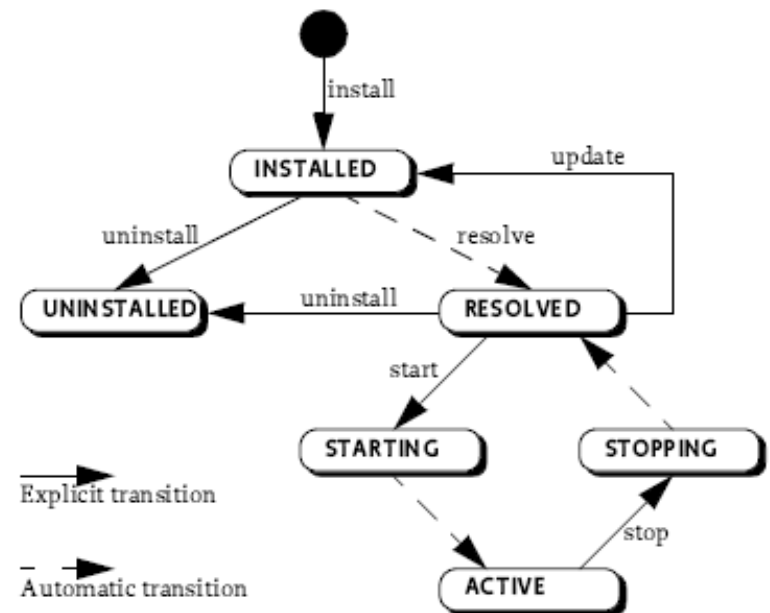
Export-Package: Package I, Package II

Import-Package: Package 1



The Lifecycle of a Bundle

- Management: Install / Start / Stop / Uninstall bundles at runtime.
- OSGi keeps track of dependencies
- What happens if a Bundle is uninstalled?



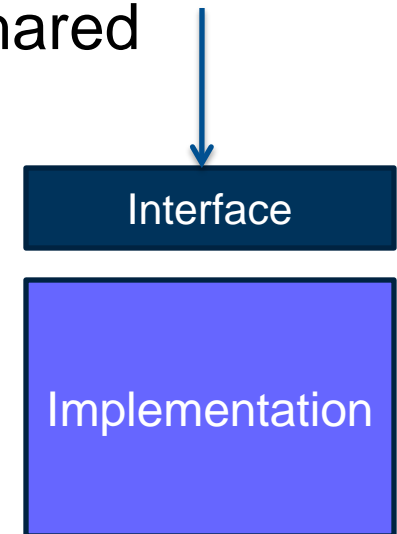
From the OSGi specifications

Java Classloading

- Class loader = Namespaces
 - Subtle: A class X loaded through class loader C1 is different from the same class X loaded through a different class loader C2
 - Cannot even cast: **Incompatible types!!!**
- So can we use only one class loader?
 - Class loaders keep a reference to every class ever loaded
 - Garbage collection of classes from uninstalled bundles?
- **Solution:** One class loader per bundle

Motivation for OSGi Services

- Modules are great to have common tasks factored out and to reuse them
- Not only statically, but also in the sense of “shared libraries” at runtime.
- But: Package dependencies are explicit.
 - Limits the modularity!
- **Solution: Services**
 - Idea: Separate the **interface** from the **implementation**



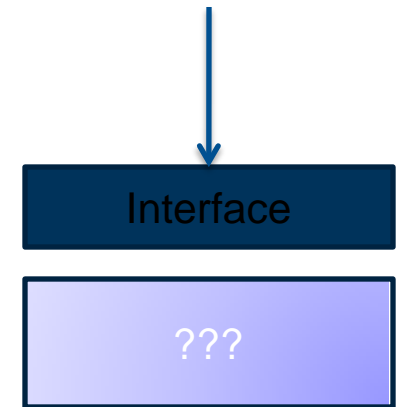
Example of a Service in OSGi

```
class MyServiceImpl implements MyService {  
  
    public void doStuff(...) {  
  
    }  
  
}
```

The application only requires knowledge about the service **interface**, the **implementation** details are hidden.

Implications

- The application has only a dependency to the service interface
 - Loose coupling
- The implementation can be exchanged, even at runtime
 - The interface is the “contract”
- Your application might be able to run even if no implementation is present
 - Optional services are possible



The Service Registry

- The OSGi framework maintains a **central service registry**
- Bundles can register their own services and retrieve services provided by other bundles
- Services can be registered with a set of **properties**
 - Additional description of the service, can be used to model constraints or do “best fit matching”



Registration of a Service in OSGi

```
package ch.ethz.test.impl;
class MyActivator implements BundleActivator {

    public void start(BundleContext context) {
        context.registerService(
            „ch.ethz.test.MyService“, new
            MyServiceImpl(), null);
    } ...
}
```

Retrieving a Service in OSGi

```
ServiceReference sref =  
    context.getServiceReference(  
        „ch.ethz.test.MyService“);  
  
if (sref != null) {  
    MyService service = (MyService)  
        context.getService(sref);  
    service.doStuff();  
}
```

Services with Properties

Service Provider

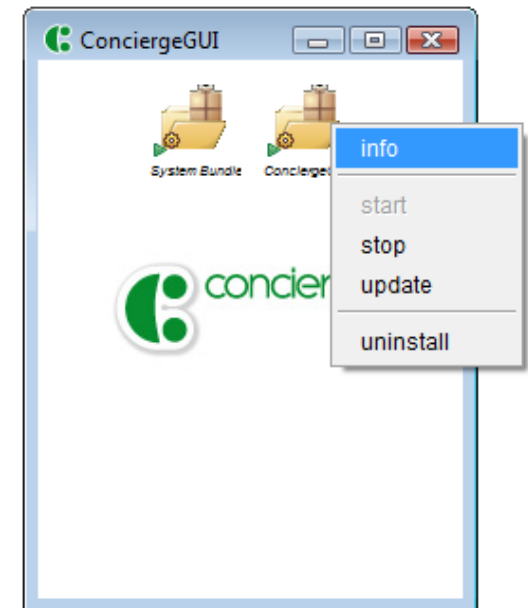
```
Dictionary properties = new  
    Hashtable();  
properties.put("color",  
    Boolean.FALSE);  
properties.put("length",  
    29);  
  
context.registerService("ch.  
    ethz.vpp.Printing", new  
    Laserjet3(), properties);
```

Service Client

```
ServiceReference[] srefs =  
    context.getServiceReferences  
    ("ch.ethz.vpp.Printing",  
    "(|(a3=true)(length>30))");
```

Concierge OSGi

- OSGi for small devices
- 85kB Jar, full OSGi R3 functionalities
- Runs on J2ME CDC 1.0 and above
- E.g., manage the software on your cell phone
- Open Source, download at <http://concierge.sourceforge.net>



[J. S. Rellermeier, G. Alonso: *Concierge - A Service Platform for Resource-Constrained Devices*. In: EuroSys 2007]

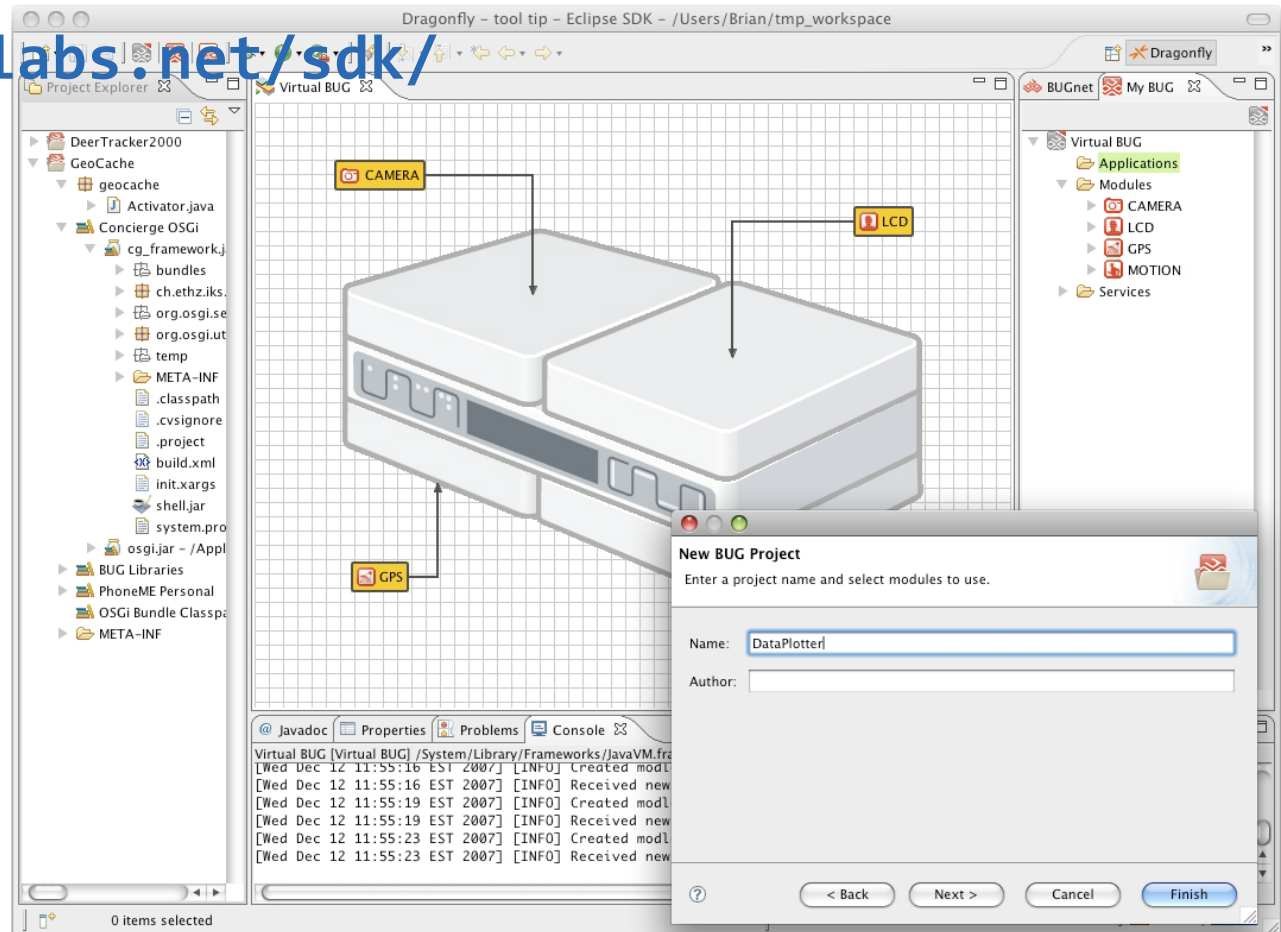
The BUG

- Device from BugLabs, New York
- “Modular Hardware”
- Runs Linux,
PhoneME, Concierge
- Hardware module drivers
map to OSGi services



Dragonfly SDK

- <http://buglabs.net/sdk/>

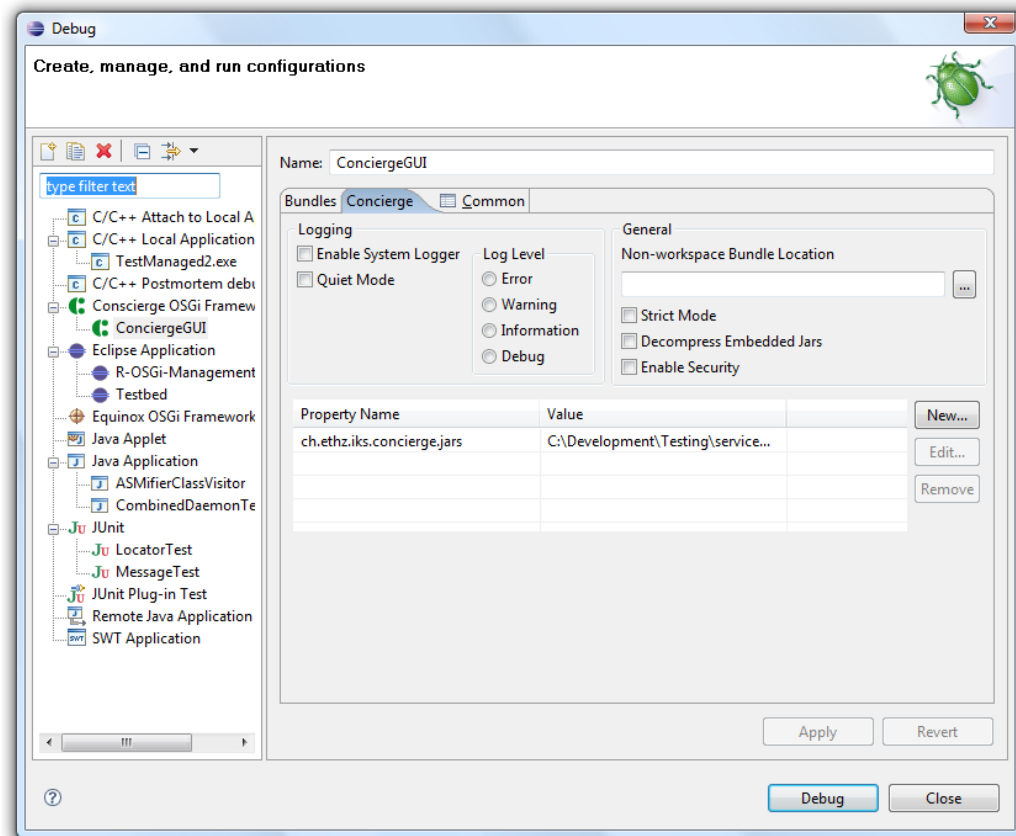


Concierge Tools for Eclipse

- Contributed by BugLabs
- Helps you to write your Bundles and Services in Eclipse
- Run and debug your services directly in Eclipse
- <http://concierge.sourceforge.net/cte/update/>



Demo



The Standard OSGi Services

- OSGi does not only describe a programming model and a framework, but also common standard services
 - Log Service
 - Event Admin
 - Permission Admin
 - Wire Admin
 - Configuration Admin
 - Many more
- OSGi Core describes the framework
- OSGi Compendium describes optional standard services



R-OSGi: Distributed OSGi Services

Jan S. Rellermeyer

Systems Group, Department of Computer Science, ETH Zurich

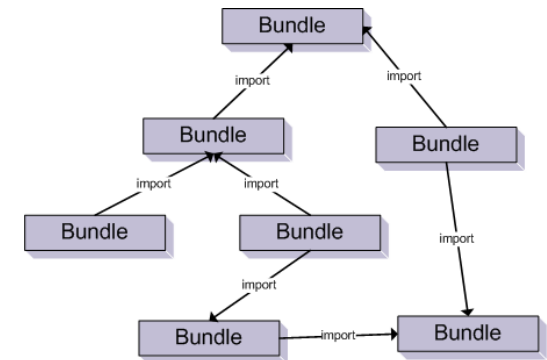


Distributed OSGi

- OSGi works only in **centralized** setups.
 - One VM, one address space, multiple applications

- Why?

- Package imports/exports
- Service Registry



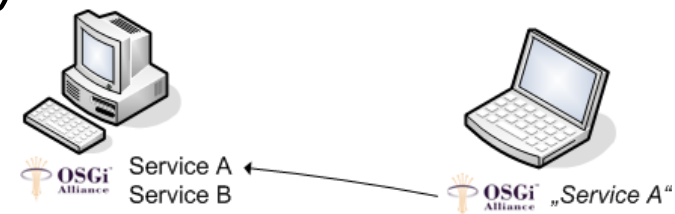
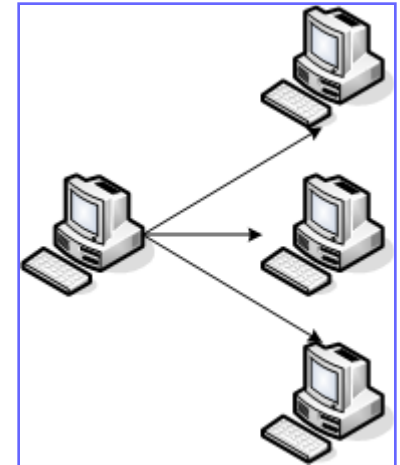
- Indeed, one can create adapters for technologies like Web Services or Jini.

- But this limits the **generality** of OSGi.

ServiceA

The R-OSGi Approach

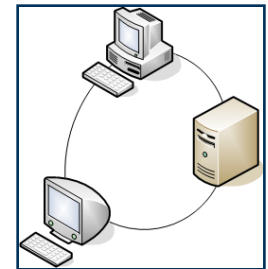
- Goal: Extending the OSGi model to distributed systems
- **Non-Invasiveness**: Leave existing applications (including the framework) untouched.
- **Transparency**: Allow the interaction with remote services in the same way as applications interact with local services.



[J. S. Rellermeier, G. Alonso, and T. Roscoe: *R-OSGi: Distributed Applications Through Software Modularization*. In: *Middleware 2007*]

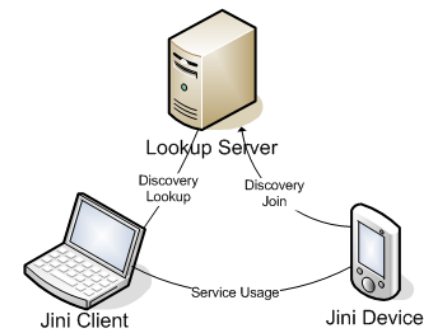
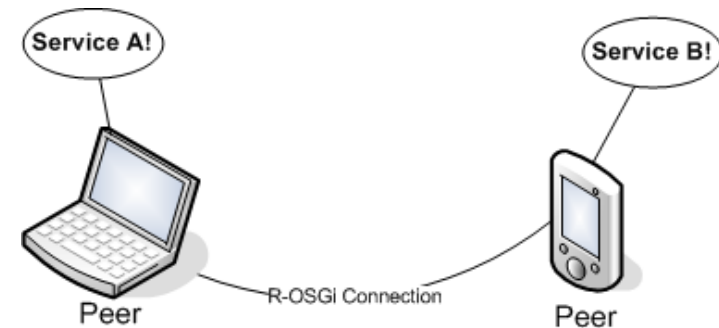
Extending the Service Registry Concept

- In some way, different service registries have to **exchange information**.
- Non-Invasiveness: Not just replacing the registry.
- How can peer A **know** that there is a matching service on peer B in an arbitrary setup?



Shared Distributed Registry

- When peer A connects to peer B, they exchange **symmetric leases**
 - A informs B about its remote services (and interest in event topics) and vice versa
- If there is a change (service added / removed / updated), the lease has to be **renewed**
 - Lease = temporary contract

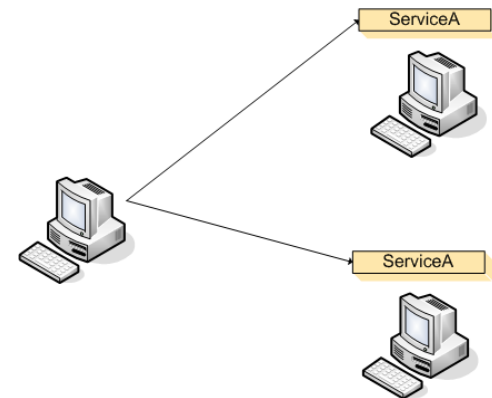


SLP: Service Discovery in the Network

- Well-established protocol, described in RFC 2608.
- Allows **service discovery** either by using a central registry server (Directory Agent) or performing multicast discovery.
- R-OSGi can use SLP
 - our own pure Java implementation jSLP
<http://jslp.sourceforge.net>
- Gives the system “**hints**” to whom to connect to

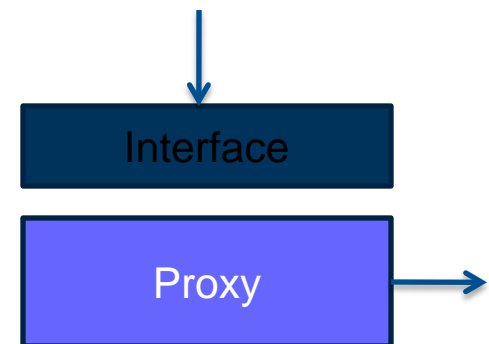
Remote Services on Demand

- Problem: There might be too many services to “import” them all into the local service registry.
- Solution: Let something in the system state the **demand**.
- If there is a demand for an external, discovered service, “import” it into the service registry.
- Import it? How?

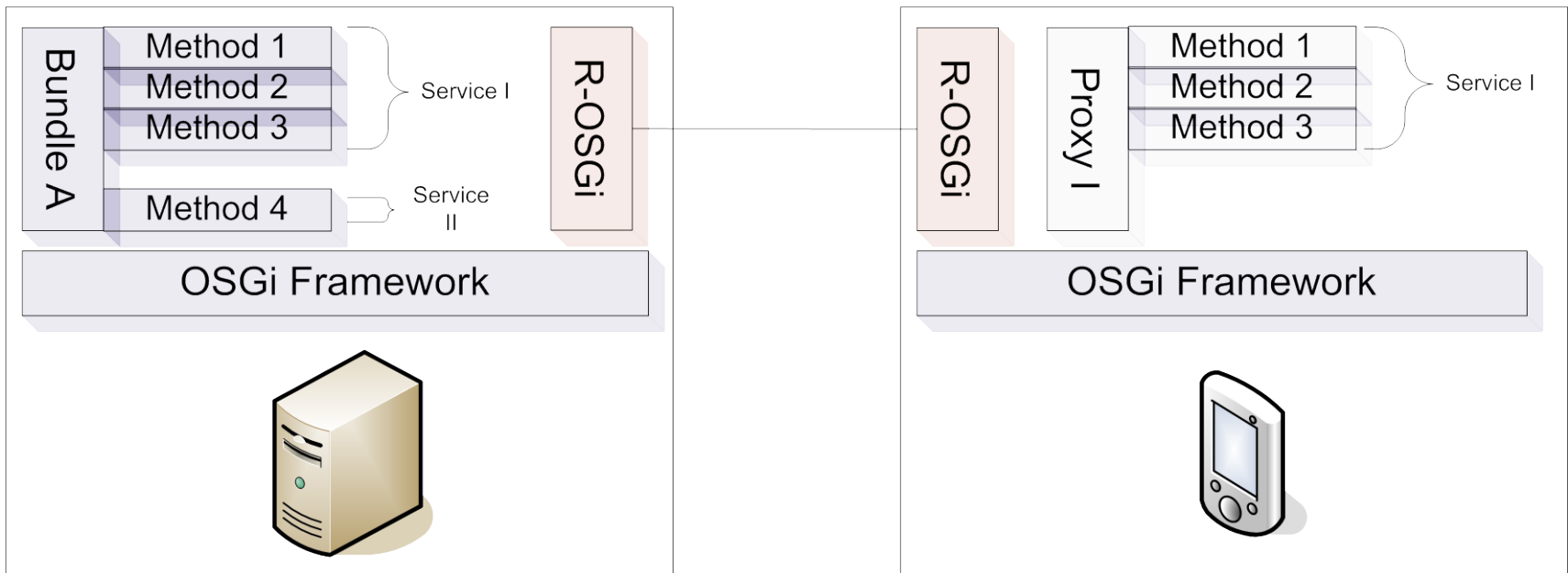


Proxy Generation

- Remember: All that a client has to know about a service is the service **interface**.
- Solution: Let the host framework transmit the service interface to the client framework and create a proxy **on the fly**. (byte code engineering)
- Register this proxy under the original service interface.
 - The proxy will look equivalent to a client.



The Big Picture

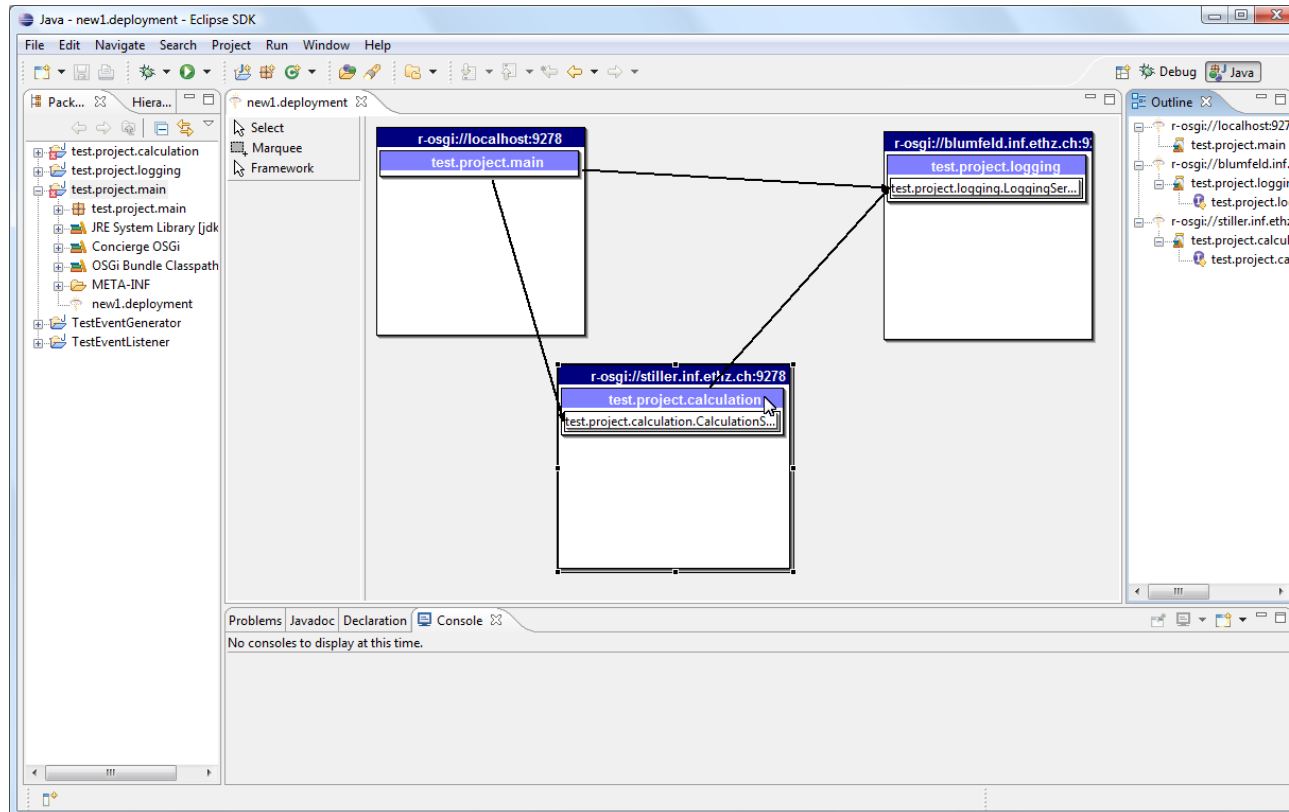


Type Injection

- What if your service uses custom classes as arguments of method calls?
- In this case, the interface cannot be resolved without these classes.
- Solution: The R-OSGi instance on the host framework prepares the “**type injections**”.
- The injections are materialized in the proxy bundle.



Example: Drag and Drop Distribution



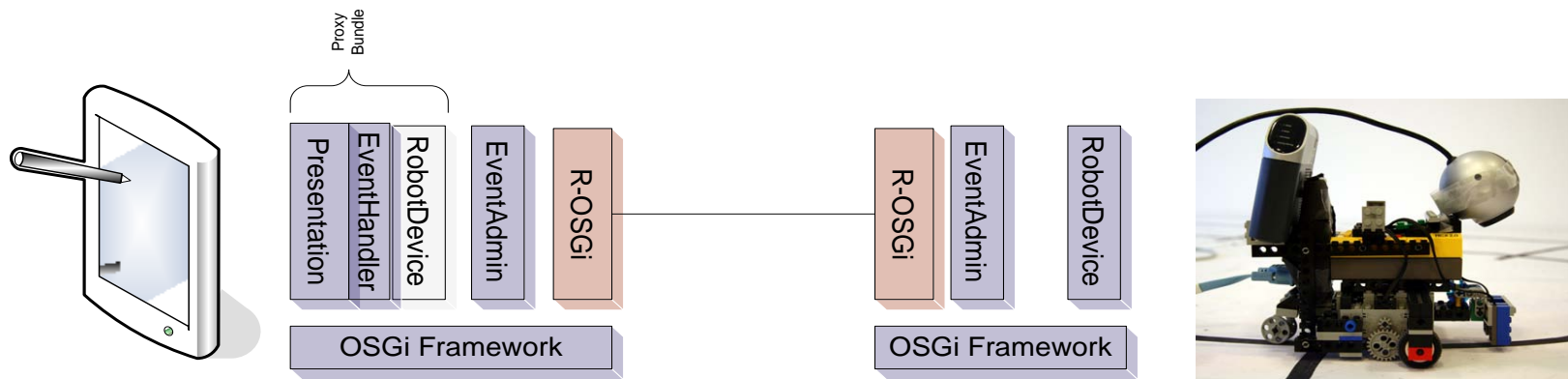
[J.S. Rellermeier, G. Alonso, T. Roscoe: *Ready for Distribution? Turning Modular into Distributed Applications with the R-OSGi Deployment Tool*. Demo at OOPSLA 07].

Example: Service Presentations (AlfredO)

- Use OSGi services to interact with ambient devices.
- Is it possible to interact with an entirely unknown service?
- Idea: The **universal remote control**.
 - Services can have presentations attached.
 - A general purpose client UI displays these attached presentations.



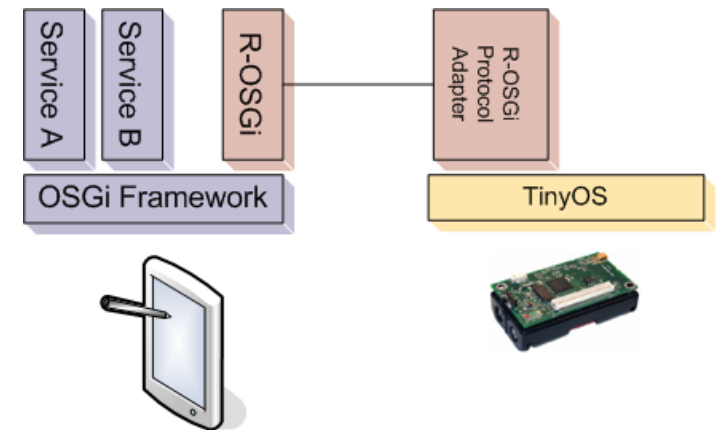
Demo: R-OSGi Presentations and AlfredO



[J. S. Rellermeier, O. Riva, G. Alonso: *AlfredO: An Architecture for Flexible Interaction with Electronic Devices*. In: *Middleware 2008*]

Sensor Nodes as R-OSGi Services

- TMote Sky
 - TI MSP430F1611 microcontroller at up to 8 MHz
 - 10k SRAM, 48k Flash + 1024k serial storage
 - 250kbps 2.4 GHz Chipcon CC2420 IEEE 802.15.4 Wireless Transceiver
- Cannot even run an OS
 - Runs TinyOS
- But it can be an R-OSGi service...



[J.S. Rellermeier, M. Duller, K. Gilmer, D. Maragkos, D. Papageorgiou, and G. Alonso: *The Software Fabric for the Internet of Things*. In: Internet of Things 2008].

Cirrostratus: The Virtual OSGi Framework

■ OSGi on the cloud

- Have a network full of machines running OSGi
- Don't care where they are
- Don't care where bundles are installed
- Don't care where services are provided
- Access them from anywhere

Bundles and services
are becoming virtual

Access them
transparently

■ Fluid OSGi

- Have a replica where you need it
- Read any / write any
- Mobility
- Dependability

From a peer's
perspective, services
"flow" through the
network

Getting involved in OSGi

- OSGi is a hot topic in the industry
- Eclipse, Apache, Sun Glassfish, IBM WebSphere and others, Oracle BEA product suite, Siemens, BMW cars, ...
- Want to get involved?
 - Play around with our software, it's all open source.
 - Start a Semester/Lab/Master's Thesis or a Research Project with our group
- OSGi DevCon @ Jazoon, 22.06.2009

