
Entwicklung verteilter Eclipse RCP & OSGi Anwendungen mit dem Eclipse Communication Framework (ECF)

Markus Alexander Kuppe
Oktober 2010

§ 1 Tag:

- Vormittags:
 - § Vortrag + Übung verteilte Anwendungen mit RMI
- Mittagspause: 12:00 - 13:00 Uhr
- Nachmittags:
 - § Vortrag + Übung verteilte Anwendungen mit JMS

§ 2 Tag:

- Vormittags:
 - § Vortrag + Übung verteilte Anwendungen mit ECF Distributed OSGi
- Mittagspause: 12:00 - 13:00 Uhr
- Nachmittags:
 - Vortrag + Übung verteilte Anwendungen mit ECF Distributed OSGi
 - Advanced Concepts

1. Bestandsaufnahme gestrige RMI-Lösung
2. Das Eclipse Communication Framework und „Remote OSGi“
 1. Remote Services
 2. ECF Discovery
 3. *Fallacies of distributed computing*
 4. Provider Architektur
 1. R-OSGi vs Generic
 2. (DNS-SD)
 5. Blackbox-Bundles
 1. R-OSGi Smart Serialization
 2. R-OSGi Smart Proxies
 6. Asynchronous
 1. Callbacks & Futures

RMI Remote Interface & throws RemoteException

```
package de.clwps.winterschool.domain.kunde.service;

import de.clwps.winterschool.domain.kunde.fachwerte.Kundennummer;
import de.clwps.winterschool.domain.kunde.material.Adresse;
import de.clwps.winterschool.domain.kunde.material.Kunde;

public interface IKundenService {

    Kunde[] listKunden();

    boolean existiertKunde(Kundennummer kundennummer);

    ...
}
```

Verlust der POJOs!

```
package de.clwps.winterschool.domain.kunde.service;

import java.rmi.Remote;
import java.rmi.RemoteException;

import de.clwps.winterschool.domain.kunde.fachwerte.Kundennummer;
import de.clwps.winterschool.domain.kunde.material.Adresse;
import de.clwps.winterschool.domain.kunde.material.Kunde;

public interface IKundenService extends Remote {

    Kunde[] listKunden() throws RemoteException;

    boolean existiertKunde(Kundennummer kundennummer) throws RemoteException;
}
```

RMI Parameter-Objekte (Serialization)

```
package de.clwps.winterschool.domain.kunde.material;

public class Adresse {

    private String _strasse;
    private int _plz;
    private String _hausnummer;
    private String _ort;
    ...
}
```

Verlust der POJOs!

```
package de.clwps.winterschool.domain.kunde.material;

import java.io.Serializable;

public class Adresse implements Serializable {

    private static final long serialVersionUID = -6702479686629330366L;

    private String _strasse;
    private int _plz;
    private String _hausnummer;
    private String _ort;
    ...
}
```

```
package de.c1wps.winterschool.ui.kundenlister;

import java.rmi.RemoteException;

public class KundenListView extends ViewPart {
    private void refreshInput() {
        if (service != null) {
            try {
                Kunde[] kunden = service.listKunden();
                viewer.setInput(kunden);
            } catch (RemoteException e) {
                //do something useful in case of problems
                e.printStackTrace();
            }
        }
        ...
    }
}
```

try/catch der RemoteException in der Domain Logik

RMI Parameter-Objekte (Serialization) fort.

- „implements Serializable“ als Marker-Interface
- Komplexe Typen
 - „transient“ keyword
 - java.io.Externalizable (extends Serializable)

```
package de.clwps.winterschool.domain.kunde.material;
import java.io.Externalizable;
import java.io.IOException;
import java.io.ObjectInput;
import java.io.ObjectOutput;
import java.io.Serializable;

public class Adresse implements Serializable, Externalizable{
    private static final long serialVersionUID = -6702479686629330366L;

    private transient Kunde _kunde;
    private transient Kunde _worker;
    private transient Kunde _geodata;

    void readExternal(ObjectInput arg0) throws IOException,
        ClassNotFoundException {
        //deserialize
    }

    public void writeExternal(ObjectOutput arg0) throws IOException {
        //serialize manually
    }

    ...
}
```

Mehr auf <http://java.sun.com/developer/technicalArticles/Programming/serialization/>

RMI Client & Proxy und ServiceProxy

```
package de.clwps.winterschool.serviceproxy.kunde_rmi.internal;

import java.rmi.registry.LocateRegistry;
import java.rmi.registry.Registry;
import org.osgi.framework.BundleActivator;
import org.osgi.framework.BundleContext;
import de.clwps.winterschool.domain.kunde.service.IKundenService;

public class KundenProxyActivator implements BundleActivator {

    public void start(BundleContext context) throws Exception {

        Registry registry = LocateRegistry.getRegistry("localhost", 1088);
        IKundenService kundeService =
            (IKundenService) registry.lookup(IKundenService.class.getName());

        context.registerService(IKundenService.class.getName(), kundeService, null);
    }
    ...
}
```

- *LocateRegistry.getRegistry(...)* - Woher stammen die Parameter?
- Einen *ServiceProxyActivator* pro Service
 - Skaliert nicht!

RMI Server und ServiceProvider

```
package de.clwps.winterschool.service.serviceprovider_rmi;

import java.rmi.RemoteException;
import java.rmi.registry.LocateRegistry;
import java.rmi.registry.Registry;
import java.rmi.server.UnicastRemoteObject;

import de.clwps.winterschool.domain.kunde.service.IKundenService;

public class ServiceProviderRMI {

    private Registry registry;
    private int port = 1088;

    public ServiceProviderRMI() throws RemoteException {
        registry = LocateRegistry.createRegistry(port);
    }

    public void bindKundenService(IKundenService kundenService) {
        try {
            IKundenService remoteKundenService = (IKundenService)
                UnicastRemoteObject.exportObject(kundenService, port);
            registry.rebind(IKundenService.class.getName(), remoteKundenService);
        } catch (RemoteException e) {
            e.printStackTrace();
        }
    }
}
```

• *bind*Service(...)* für jeden Service nötig

```
Manifest-Version: 1.0
Bundle-ManifestVersion: 2
Bundle-Name: Serviceprovider_rmi
Bundle-SymbolicName: de.clwps.winterschool.service.serviceprovider_rmi
Bundle-Version: 1.0.0.qualifier
...
DynamicImport-Package: *
```

- Classloader Legacy Hacks notwendig
 - Portabel auf allen OSGi Runtimes?

1. *Serializable*, *Remote*, *throws RemoteException* kontaminieren die POJOs mit technischen Details der Remote-Technologie
 - Austausch der Remote-Technologie ist kaum bis gar nicht möglich
 - Lösung: Abstrahieren von *java.rmi.Remote*?
 - Wann ist *throws RemoteException* sinnvoll?
2. *try/catch RemoteException* in Service Nutzern/Domain Logik (Kontaminierung)
3. Ein *ServiceProxy* und *ServiceProvider* Bundle für:
 - Jeden **Service**
 - Jede **Remote-Technologie**
4. RMI-Registry muss global bekannt sein
 - Lösung: In XML Konfiguration auslagern?
5. (nicht portable) Classloading Tricks an OSGi vorbei
6. Blockierendes Verhalten synchrone Kommunikation

=> „Remoting“ ist nicht transparent!

1. Bestandsaufnahme gestrige RMI-Lösung

2. Das Eclipse Communication Framework und „Remote OSGi“

1. Remote Services

2. ECF Discovery

3. *Fallacies of distributed computing*

4. Provider Architektur

1. R-OSGi vs Generic

2. (DNS-SD)

5. Blackbox-Bundles

1. R-OSGi Smart Serialization

2. R-OSGi Smart Proxies

6. Asynchronous

1. Callbacks & Futures

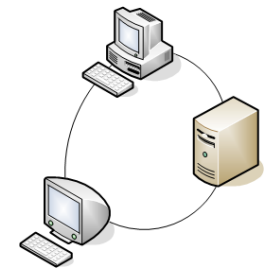
·Communication platform of Eclipse

- Around since 2004 (ECF 3.4)
- Mature use/contributor base
- Eclipse Runtime (rt) project



·Mission

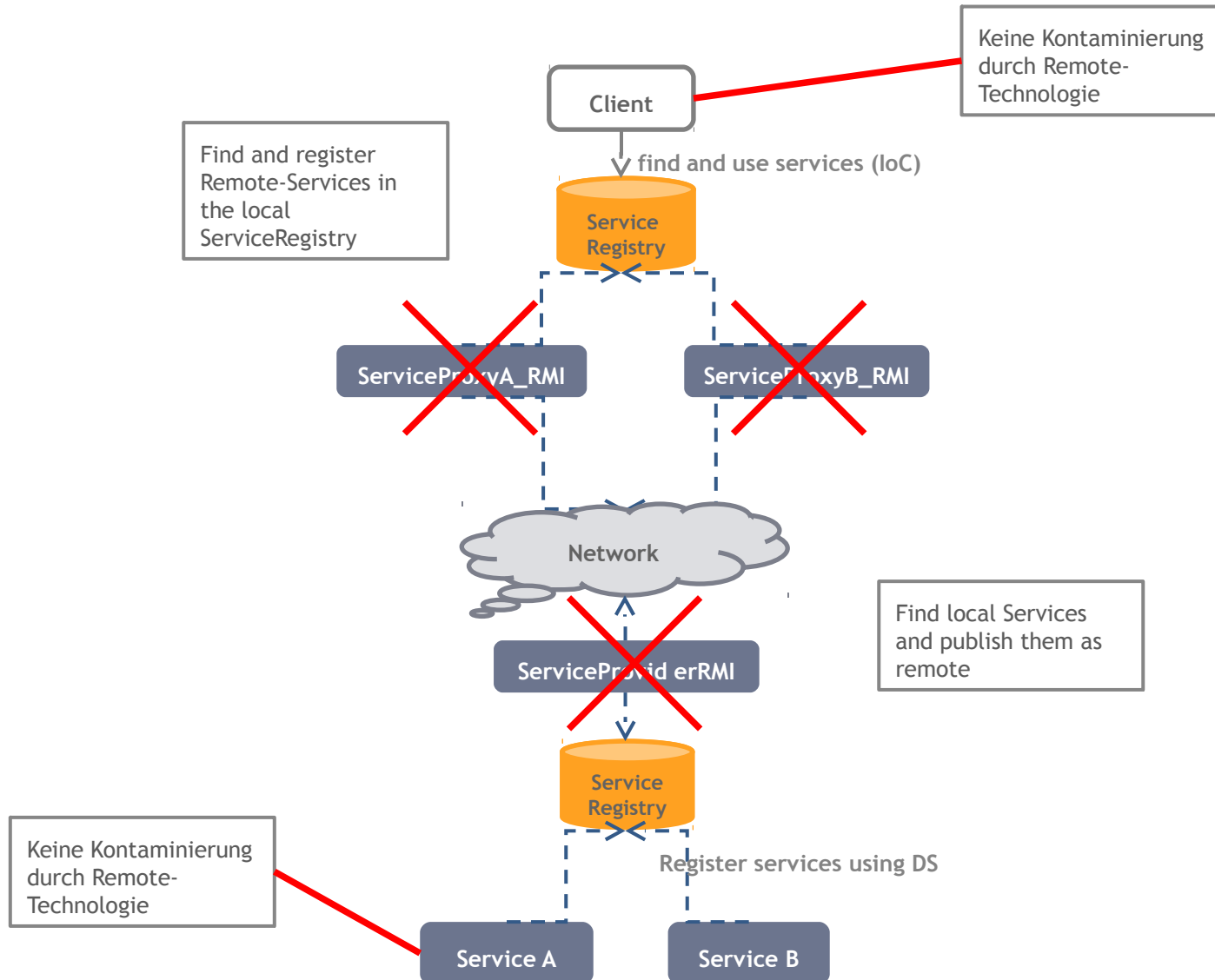
- Support team and community collaboration
- In combination with the Eclipse IDE
 - Shared editing, file transfer, messaging
- **As an interprocess communication platform for OSGi apps**
 - **e.g., service discovery, remote services**



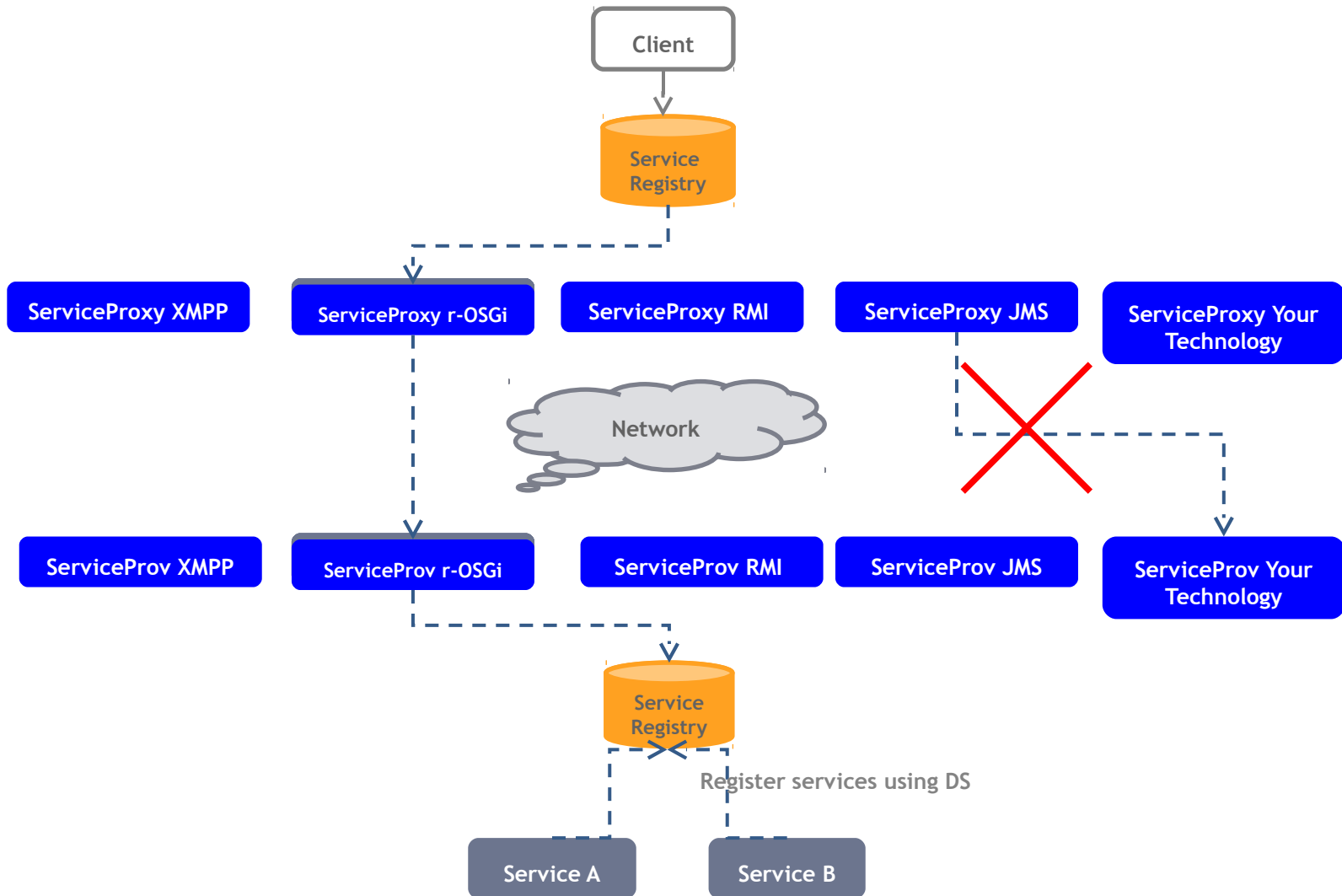
- Fall 2004: Incubated as Technology Project
 - Communications and Messaging APIs
 - Real-time collaboration applications (IM/IRC/Presence/etc)
- Spring 2007: Europa/ECF 1.0
 - Variety of APIs/Apps on APIs: i.e. Core, Presence, Filetransfer, Discovery, Remote Services
- Spring 2008: Ganymede/ECF 2.0
 - Filetransfer used by SDK/P2
 - N&N: RT Shared Editing, Bot API, UI features
 - Move to Runtime Project
- Ongoing Project Goals
 - Open Source + Open Protocol
 - Transport Independence/Interoperability – Provider Architecture
 - Support Team Collaboration in Eclipse
 - Add inter-process communications support into Equinox

- Versant Corp. for *Vitness*
- Cloudsmith Inc. for *Buckminster*
- Eclipse Platform includes *Equinox p2* which uses ECF for Install/Update download
- IRC users on irc.freenode.net, #eclipse, #eclipse-dev, #eclipse-ecf and others use the *KOS-MOS IRC bot*, which was built using the ECF Bot Framework.
- The *EPP* project includes ECF in the RCP package
- The *eConference* project
- jACT-R*, a cognitive simulation system is exploring ECF for distributed model execution
- Coffee: <http://www.coffee-soft.org/>

Architektur RMI Lösung



ECF Value Proposition



ECF Remote OSGi

„Silver Bullet“ der verteilten Entwicklung ;-)

1. Bestandsaufnahme gestrige RMI-Lösung
2. Das Eclipse Communication Framework und „Remote OSGi“

1. Remote Services

2. ECF Discovery
3. *Fallacies of distributed computing*
4. Provider Architektur
 1. R-OSGi vs Generic
 2. (DNS-SD)
5. Blackbox-Bundles
 1. R-OSGi Smart Serialization
 2. R-OSGi Smart Proxies
6. Asynchronous
 1. Callbacks & Futures

- § Implementieren Sie eine ECF-basierte Lösung, die die Transparenz-Problem 1., 2., 3., (4.) und 5. behebt
- Vorüberlegung:
 - Wie wird das *ServiceProvider* Bundle auf der „Provider“-Seite ersetzt, damit Kunde/Konto-Service für das „Remoting“ gekennzeichnet sind?
 - Wie wird die „Endpoint“-Konfiguration (4.) auf der „Consumer“-Seite abgelegt?
- § Vorbedingung:
- Workspace mit Buckminster aufsetzen

Aufgabe 1 - ECF Silver Bullet

Remote Service Properties (programmatisch)

```
package de.clwps.winterschool.service.kunde;
import java.util.Properties;

import org.osgi.framework.BundleActivator;
import org.osgi.framework.BundleContext;

import de.clwps.winterschool.domain.kunde.service.IKundenService;

public class KundenServiceActivator implements BundleActivator {

    public void start(BundleContext context) throws Exception {
        Properties props = new Properties();
        props.put("service.exported.interfaces", "*");
        context.registerService(
            IKundenService.class.getName(), new KundenServiceMemImpl(), props);
    }
    ...
}
```

- Lediglich eine „Marker-Property“ wird zur Kennzeichnung eines Remote-Services genutzt
 - Invasiv, da im BundleActivator?
 - Service-Designer trifft nicht immer die Entscheidung, ob ein Service remotet wird

Aufgabe 1 - ECF Silver Bullet

Endpoint description (Consumer-Seite)

```
<?xml version="1.0" encoding="UTF-8"?>
<service-descriptions xmlns="http://www.osgi.org/xmlns/sd/v1.0.0">
  <!-- IKontoService -->
  <service-description>
    <provide
      interface="de.clwps.winterschool.domain.konto.service.IKontoService"/>
    <property name="ecf.sp.cid">ecftcp://localhost:3282/server</property>
  </service-description>

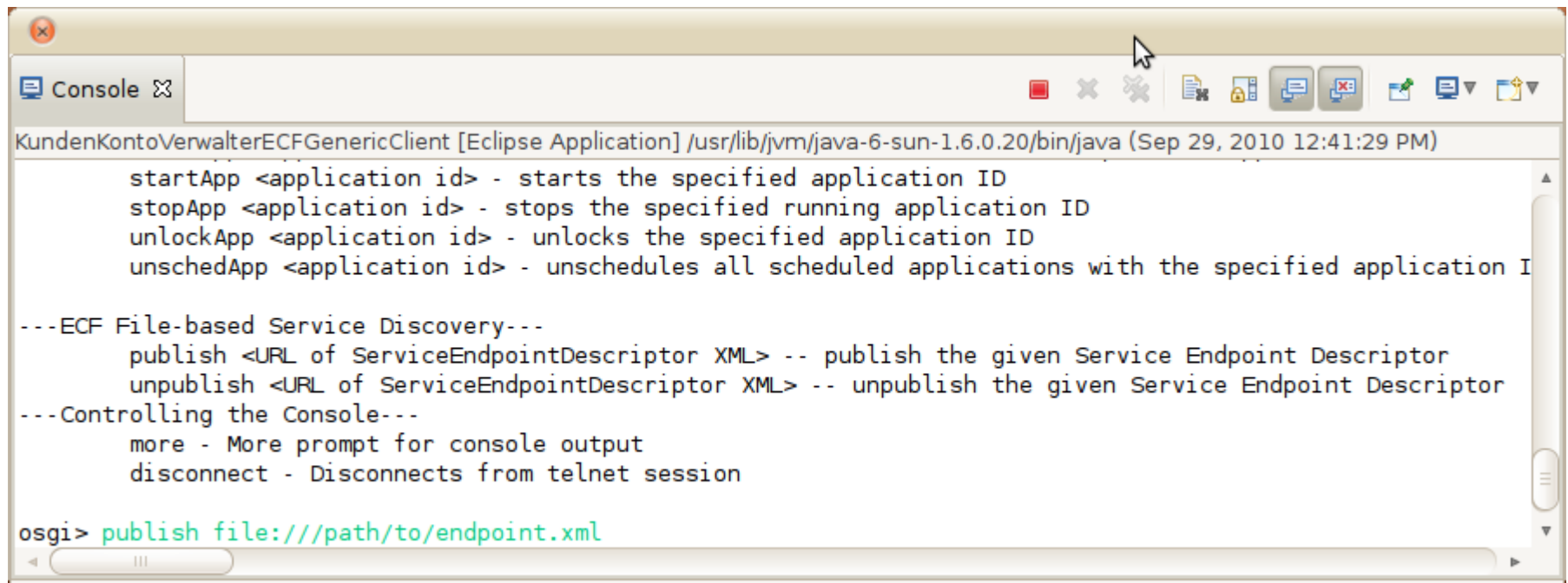
  <!-- IKundenService -->
  <service-description>
    <provide
      interface="de.clwps.winterschool.domain.kunde.service.IKundenService"/>
    <property name="ecf.sp.cid">ecftcp://localhost:3282/server</property>
  </service-description>
</service-descriptions>
```

- „*ecf.sp.cid*“ (java.net.URI) identifiziert ServiceProvider
 - [scheme:][//authority][path][?query][#fragment]
 - http://www.informatik.uni-hamburg.de:80/swt?param=foo#bar

Aufgabe 1 - ECF Silver Bullet

Endpoint description

- Eine Endpoint Description wird manuell über die Eclipse Console mit dem „File-based Service Discovery“-Kommando aktiviert
 - Alternative können Endpoint Descriptions in „*OSGI-INF/remote-service/*“ abgelegt werden und über den Bundle-Lifecycle dem OSGi-Framework bekannt gegeben werden



```
KundenKontoVerwalterECFGenericClient [Eclipse Application] /usr/lib/jvm/java-6-sun-1.6.0.20/bin/java (Sep 29, 2010 12:41:29 PM)

startApp <application id> - starts the specified application ID
stopApp <application id> - stops the specified running application ID
unlockApp <application id> - unlocks the specified application ID
unschedApp <application id> - unschedules all scheduled applications with the specified application ID

---ECF File-based Service Discovery---
publish <URL of ServiceEndpointDescriptor XML> -- publish the given Service Endpoint Descriptor
unpublish <URL of ServiceEndpointDescriptor XML> -- unpublish the given Service Endpoint Descriptor
---Controlling the Console---
more - More prompt for console output
disconnect - Disconnects from telnet session

osgi> publish file:///path/to/endpoint.xml
```

Aufgabe 1 - ECF Silver Bullet

Remote Service Properties (deklarativ)

```
<?xml version="1.0" encoding="UTF-8"?>
<scr:component xmlns:scr="http://www.osgi.org/xmlns/scr/v1.1.0" enabled="true"
immediate="true" name="kundenService">
  <implementation class="de.clwps.winterschool.service.kunde.KundenServiceMemImpl"/>

  <service>
    <provide interface="de.clwps.winterschool.domain.kunde.service.IKundenService"/>
  </service>

  <reference bind="bindKundeChangedListener"
cardinality="0..n"
interface="de.clwps.winterschool.domain.kunde.listener.IKundeChangedListener"
name="IKundeChangedListener"
policy="dynamic" unbind="unbindKundeChangedListener"/>

  <property name="service.exported.interfaces" type="String" value="*" />
</scr:component>
```

- „Marker-Property“ wird deklarativ gesetzt
 - Bei Blackbox/Glassbox Bundles nicht möglich. Alternative?
 - Extender Pattern oder ServiceHooks

Aufgabe 1 - ECF Silver Bullet

Remote Service Properties (fort.)

- „*service.exported.interfaces*“ (String+) [OSGi Service Platform Release 4.2]
 - Setting this property marks this service for export. It defines the interfaces under which this service can be exported. This list must be a subset of the types listed in the `objectClass` service property. The single value of an asterisk ('*', \u002A) indicates all interfaces in the registration's `objectClass` property and ignore the classes. It is strongly recommended to only export interfaces and not concrete classes due to the complexity of creating proxies for some type of concrete classes.
- „*service.exported.configs*“ (String+)
 - A list of configuration types that should be used to export the service. Each configuration type represents the configuration parameters for one or more Endpoints. A distribution provider should create endpoints for each configuration type that it supports. (optional)
- „*service.exported.intents*“ (String+)
 - A list of intents that the distribution provider must implement to distribute the service. Intents listed in this property are reserved for intents that are critical for the code to function correctly, for example, ordering of messages. These intents should not be configurable. (optional)

Aufgabe 1 - ECF Silver Bullet

Endpoint description

Bundle-Name: OSGi RemoteService Endpoint Dscr

Bundle-SymbolicName:

de.clwps.winterschool.endpointdscr

Import-Package:

org.osgi.framework; *version*="1.4.0,,

...

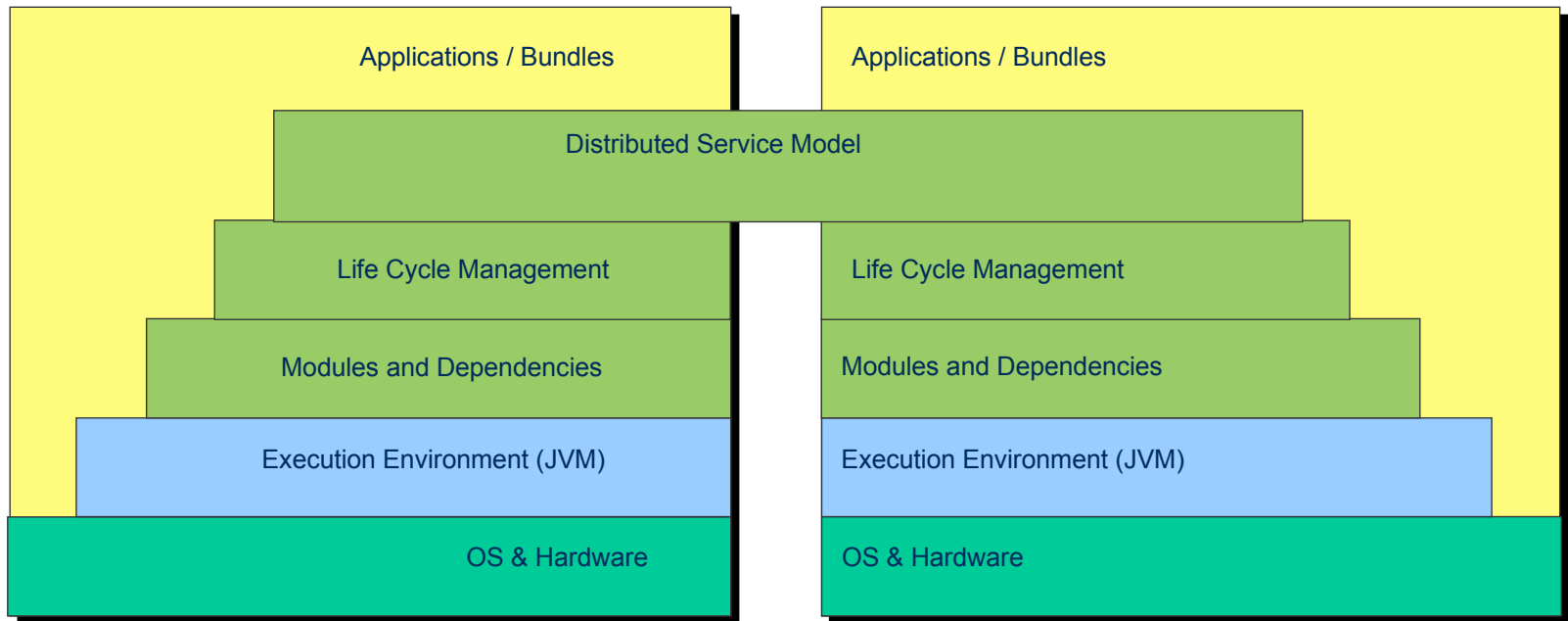
Remote-Service: **OSGI-INF/remote-service/*.xml,**
/META-INF/osgi/services.remote, \$
{java.io.tmpdir}/HelloGalileoService.xml, \$
{java.io.tmpdir}/poststart2/*.xml

Aufgabe 1 - Lessons Learned

- Fachlicher Code ist nicht mehr durch Remote-Technologie kontaminiert
 - Von Remote erben und RemoteException werfen entfällt
 - `java.io.Serializable` aber immer noch erforderlich
 - Kein try/catch in Domain Logik
- Keine ServiceProvider/ServiceProxy pro fachlichen Service
- (Keine ServiceProvider/ServiceProxy pro Remote-Technologie)
- Keine Classloading Hacks
- Endpoint description ist (zu) statisch (kümmern wir uns gleich drum)
 - Code- oder Konfigurationsänderungen, wenn Endpoint wechselt
 - Service Provider muss vor dem Service Consumer gestartet sein

ECF Silver Bullet explained - OSGi Remote Services

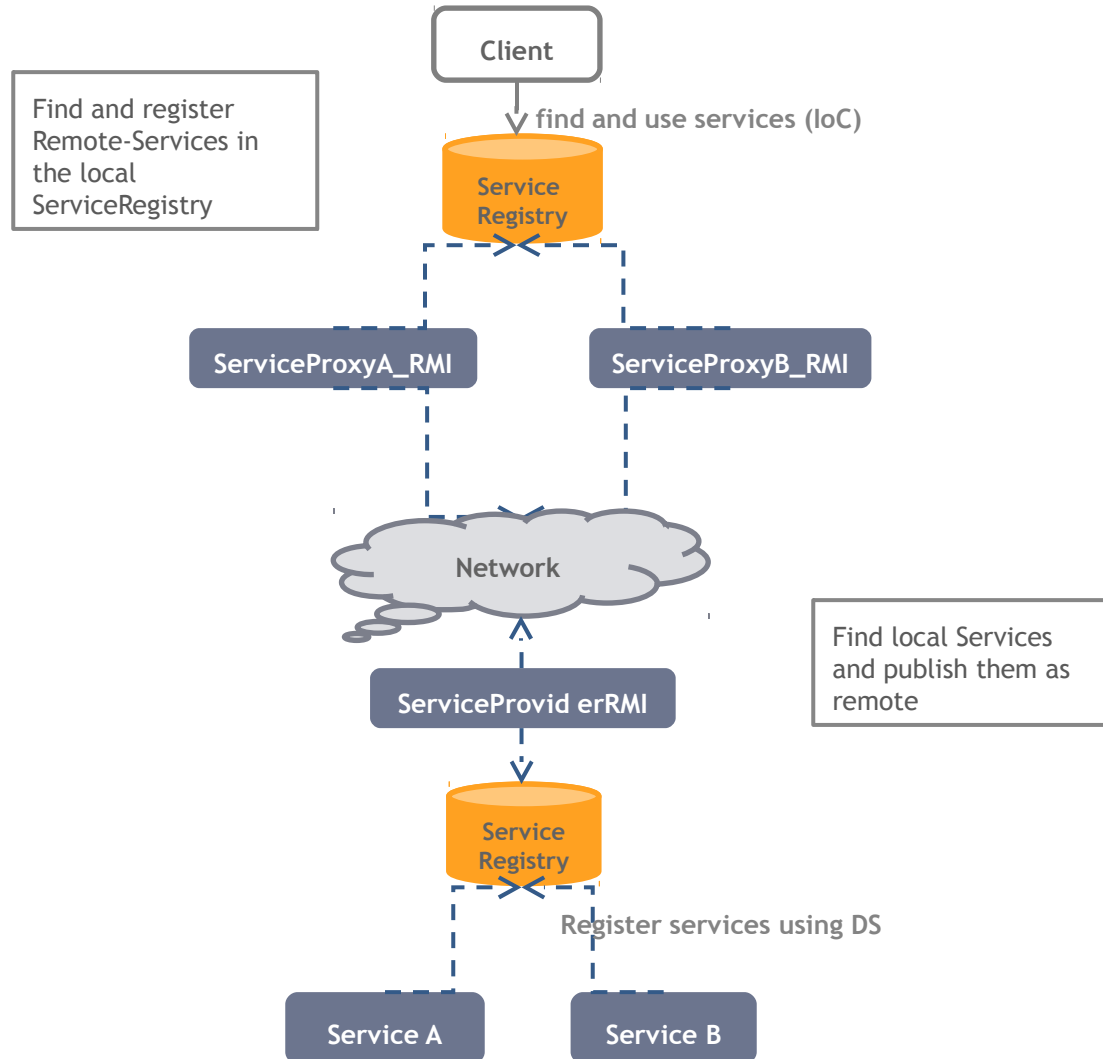
Aufgabe 1 - ECF Silver Bullet explained (fort.)



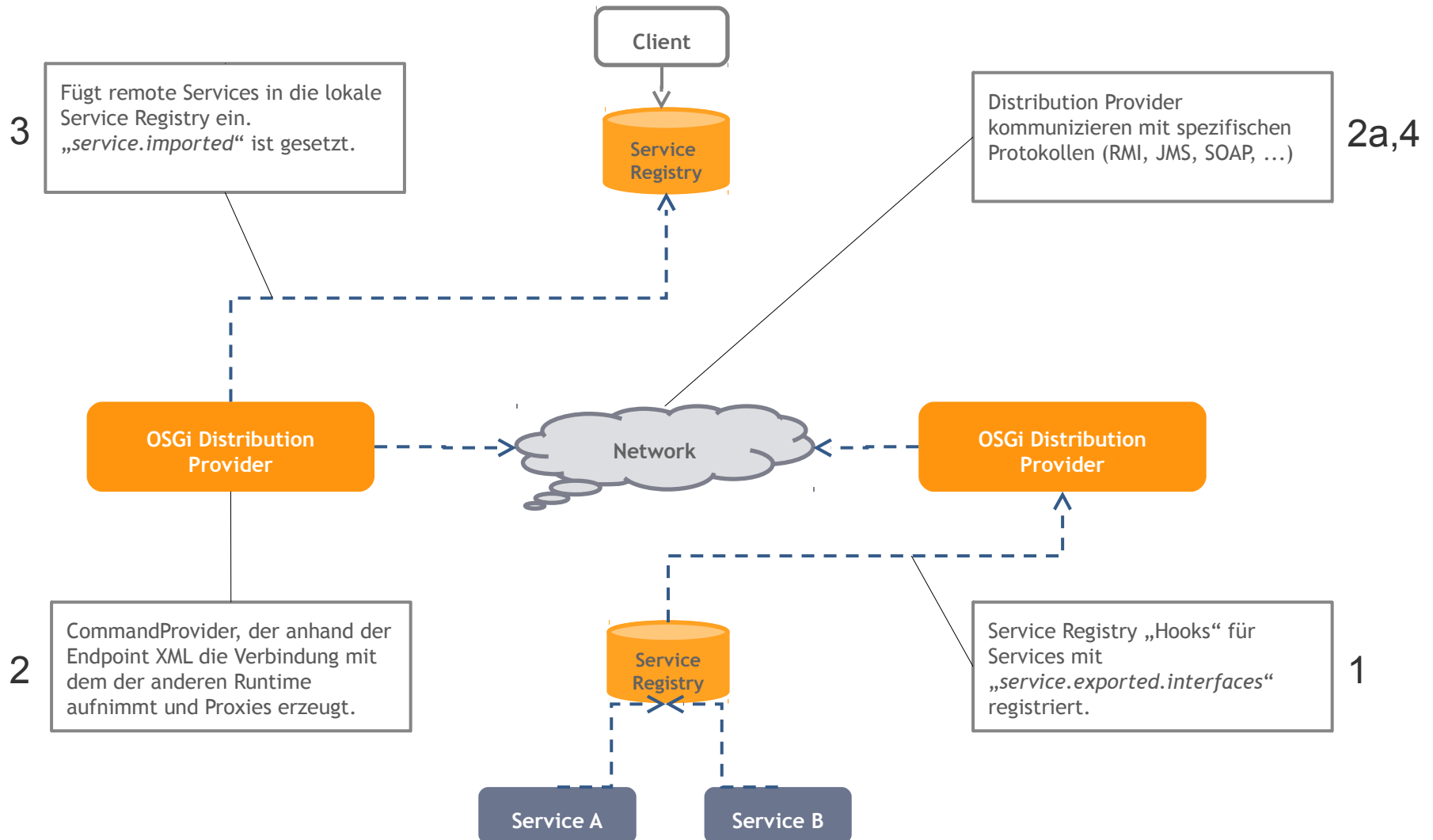
- Architektur-Richtlinie

- OSGi Programmier-Modell beibehalten
- Abstraktion von Kommunikations-Protokollen und Datenformaten
- Interoperabilität

Architektur RMI Lösung



Architektur ECF Remote Services



Aufgabe X - ECF I[Proxy|Host][Distribution|Discovery]Listener

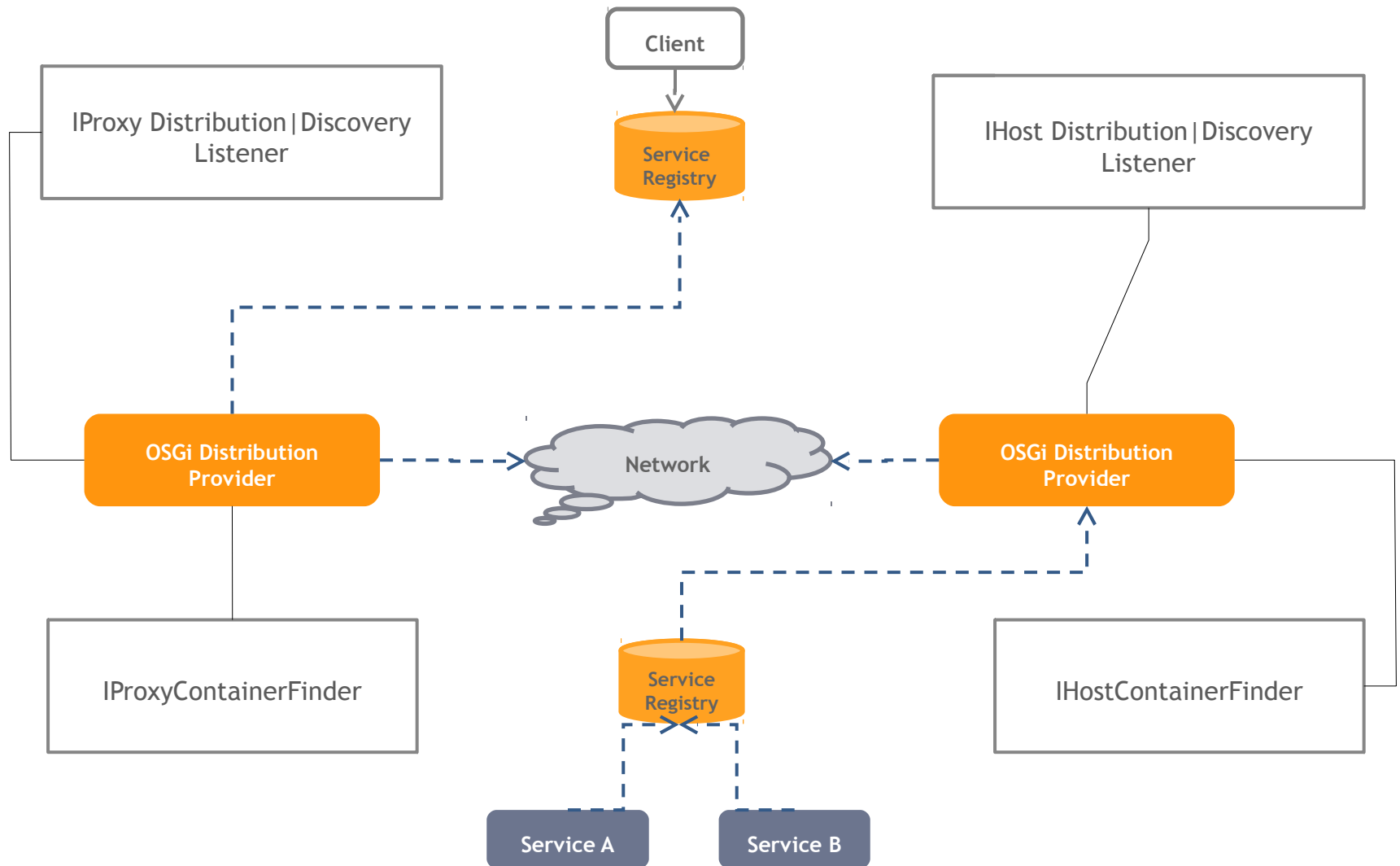
- § Implementieren Sie die unten genannten Listener-Interfaces, um sich die Vorgänge innerhalb von ECF Remote OSGi zu versinnbildlichen
- § **ServiceConsumer**
 - *IProxyDistributionListener*
 - *IProxyDiscoveryListener*
- § **ServiceProvider**
 - *IHostDistributionListener*
 - *IHostDiscoveryListener*
- § **Vorüberlegung:**
 - Wohin mit den Implementierungen, da nicht OSGi Standard, sondern ECF-spezifisch
 - => Eigene Bundle
 - Wie bekannt machen?
 - => Whiteboard Pattern

Aufgabe X - ECF I[Proxy|Host]ContainerFinder

- § Implementieren Sie die unten genannten *ContainerFinder-Interfaces* (für Wagemutige ohne von *Default[Host|Proxy]ContainerFinder* zu erben)
- § **ServiceConsumer**
 - *IProxyContainerFinder*
- § **ServiceProvider**
 - *IHostContainerFinder*
 - Vorgriff Provider Architektur

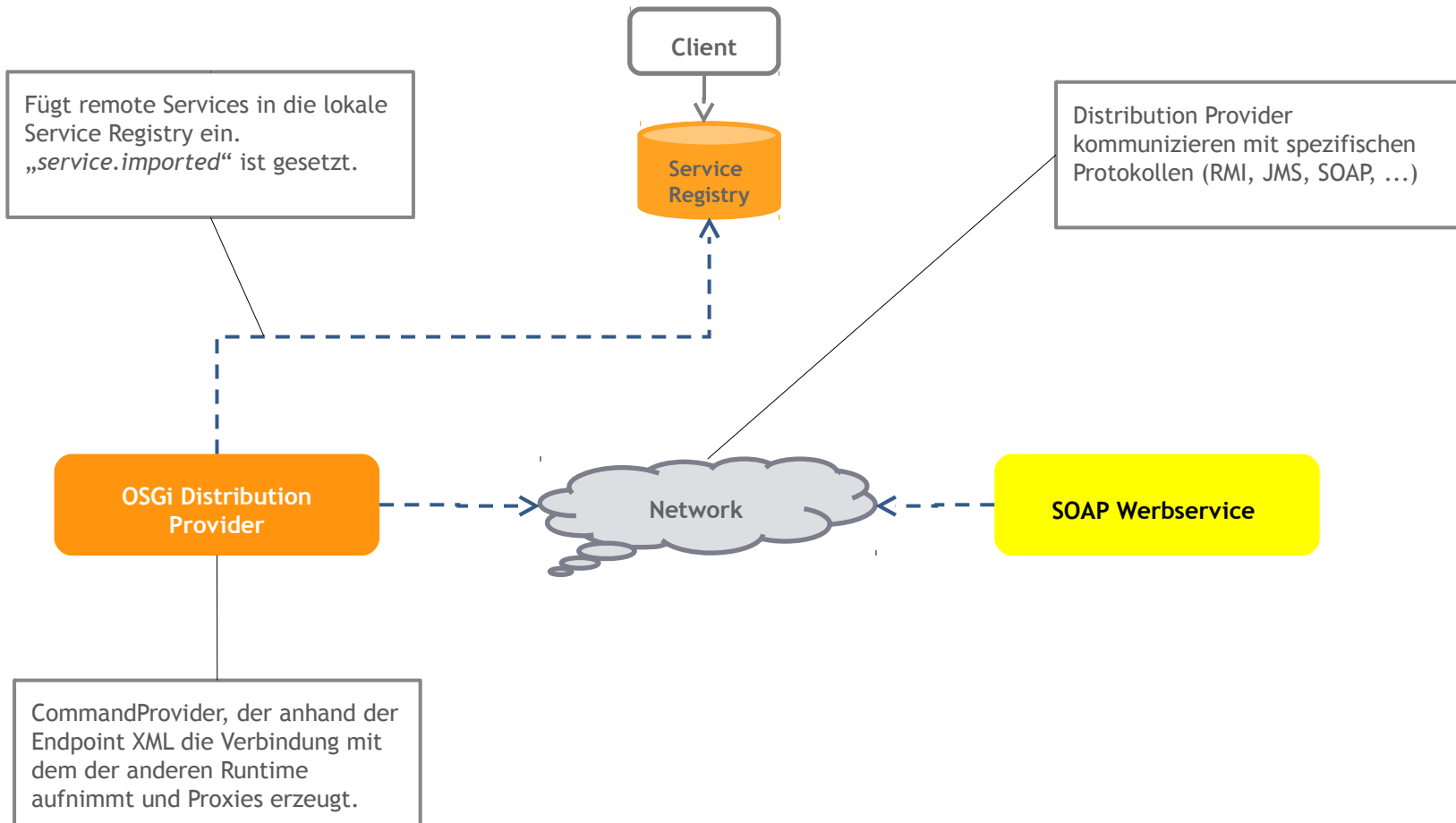
```
public MyHostContainerFinder() {  
    super(true,  
          new String[]{"ecf.generic.server"});  
}
```

Aufgabe X - ECF I[Proxy|Host]ContainerFinder



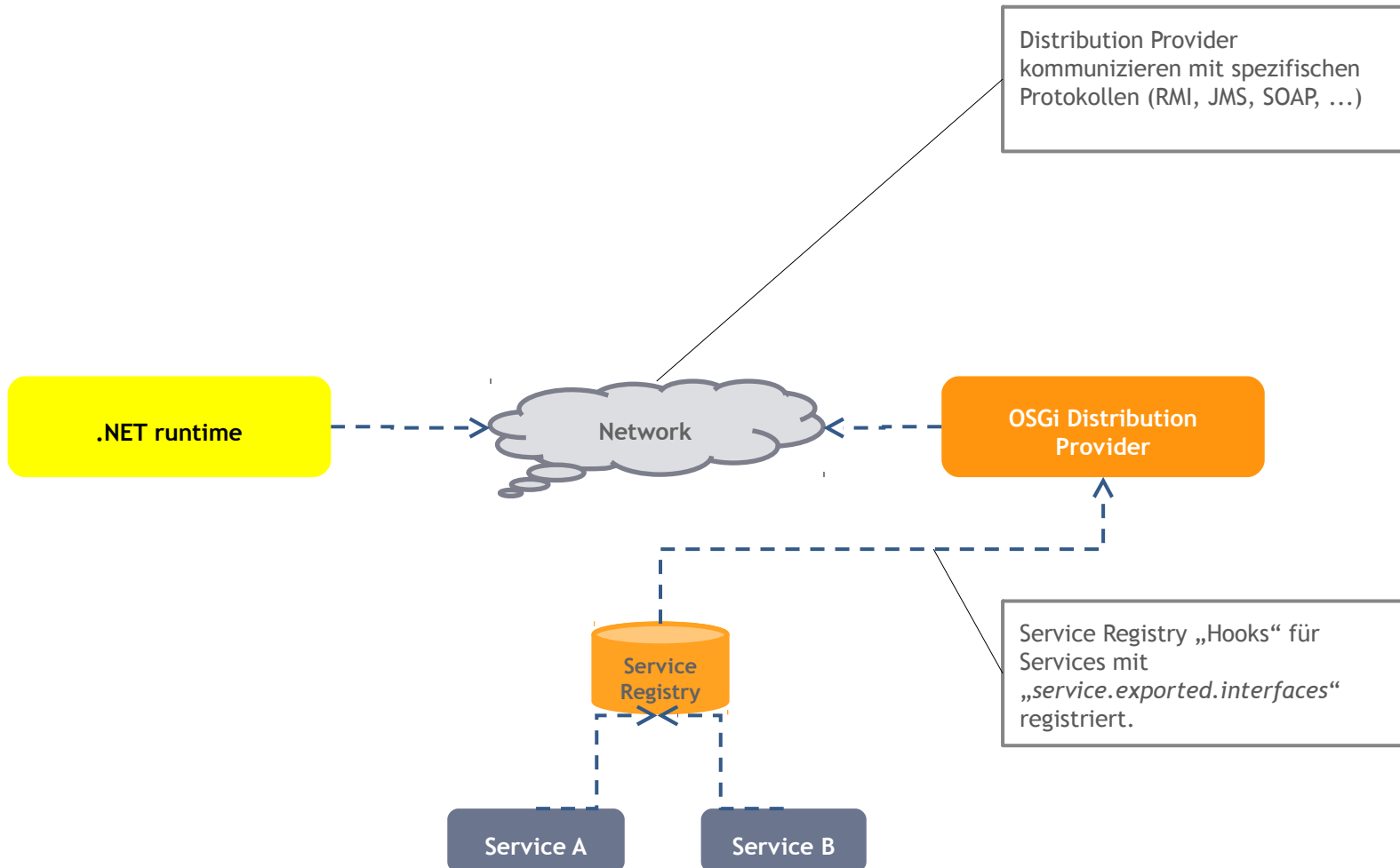
Architektur ECF Remote Services (fort.)

Non-OSGi Service Provider



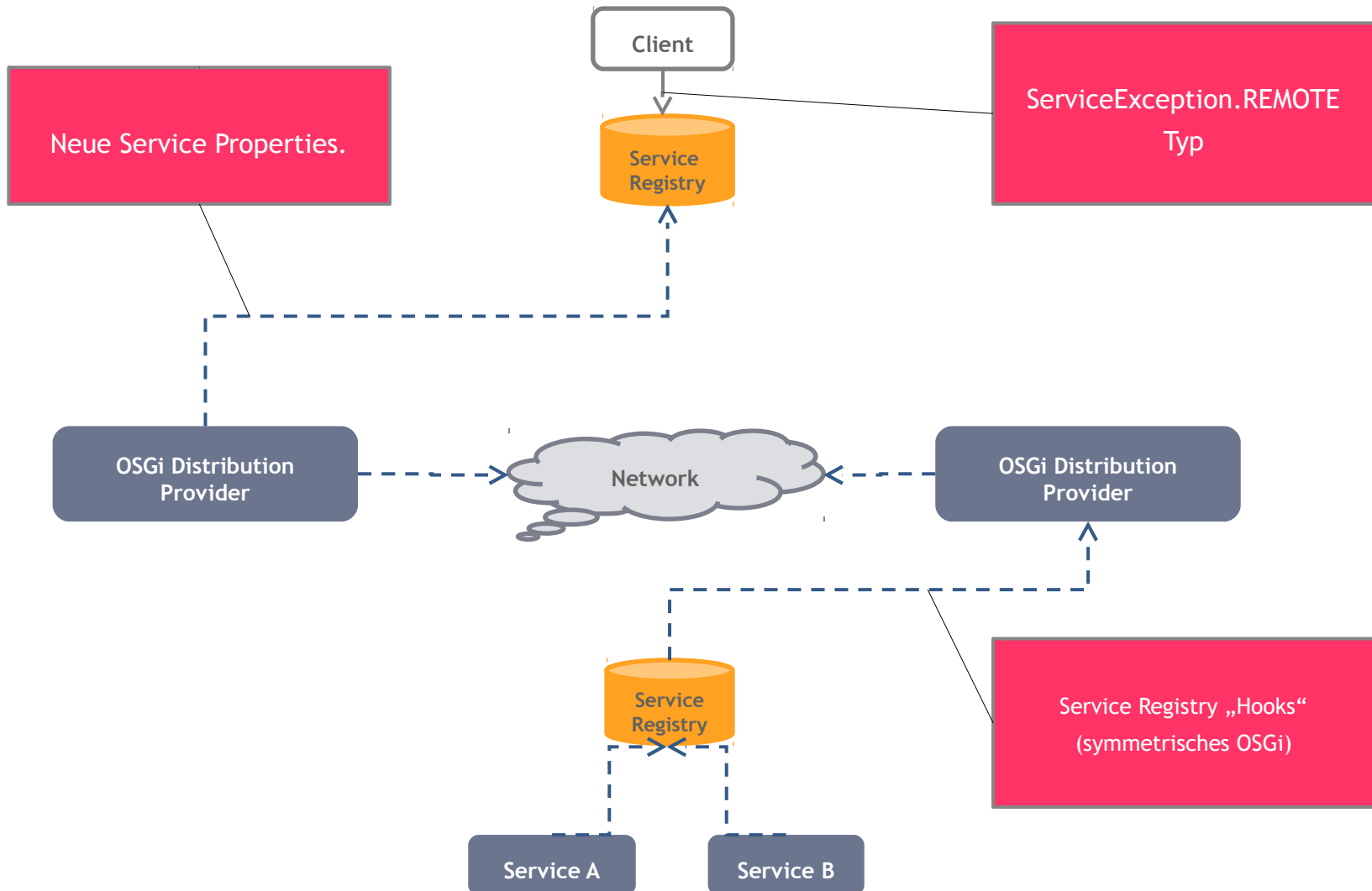
Architektur ECF Remote Services (fort.)

Non-OSGi Service Consumer

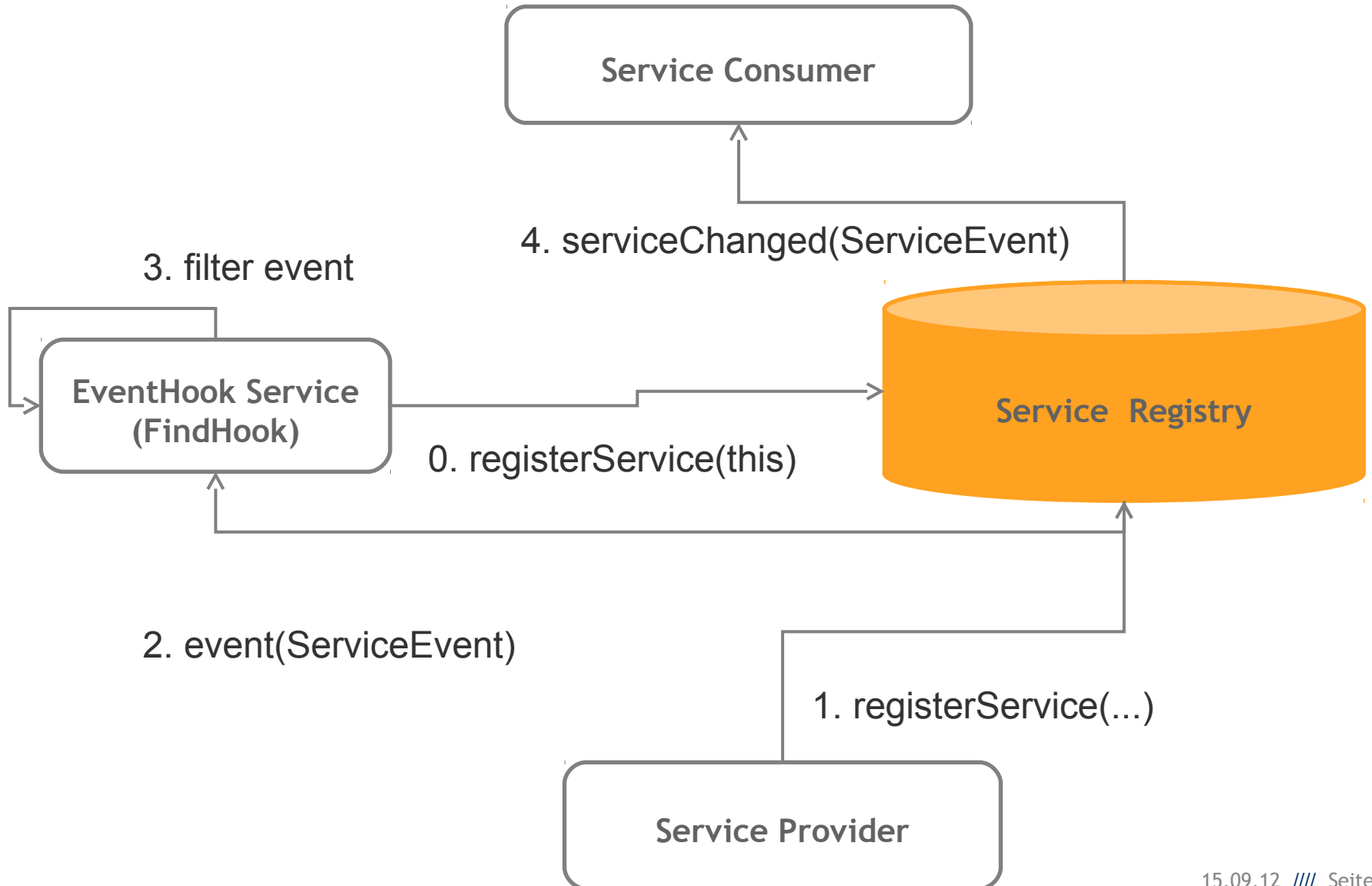


Architektur ECF Remote Services (fort.)

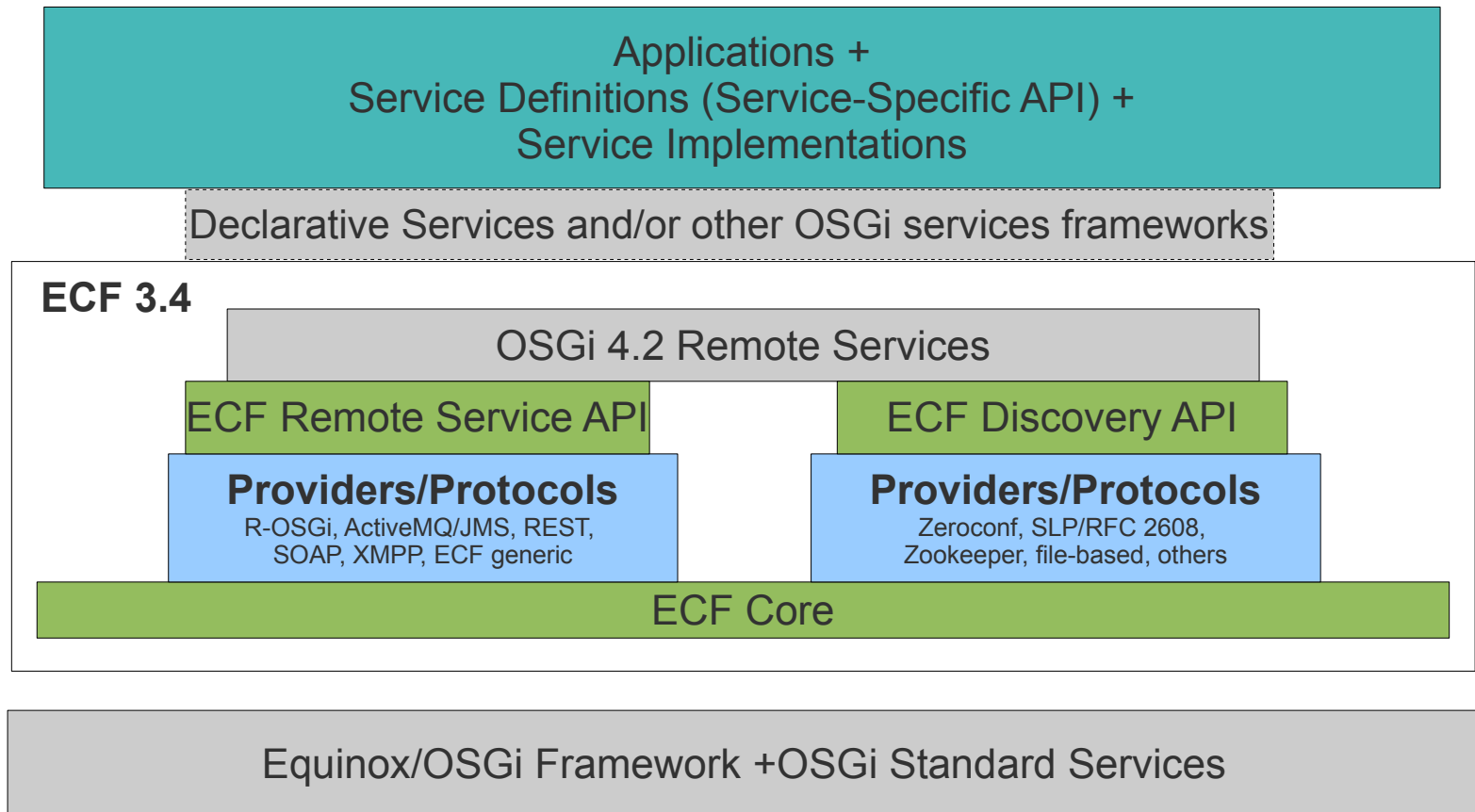
OSGi 4.2 Erweiterungen



Exkurs: Service Hooks



Architektur ECF Remote Services (fort.)



Aufgabe 1 - Lessons Learned

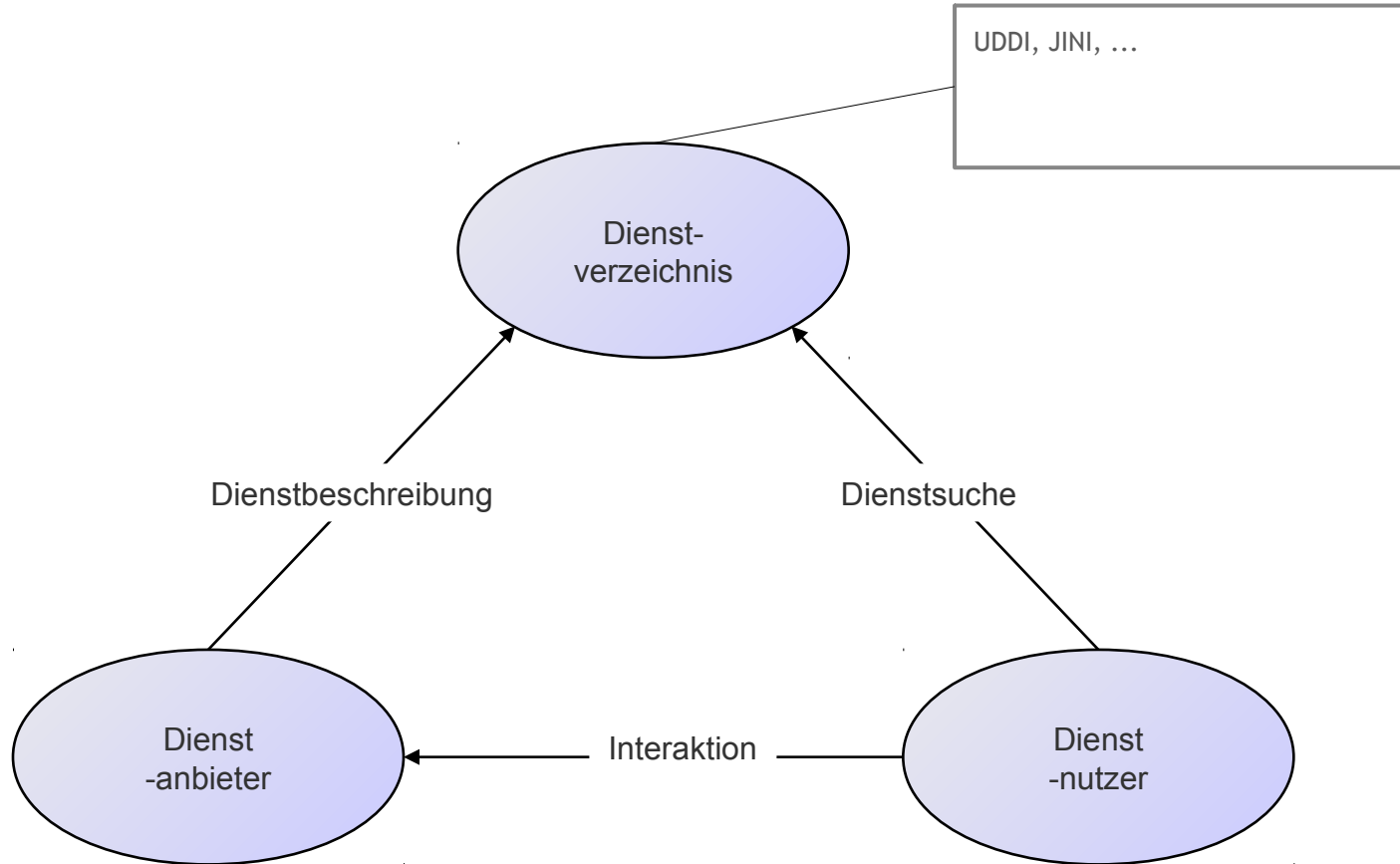
- ~~Fachlicher Code ist nicht mehr durch Remote-Technologie kontaminiert~~
 - ~~Von Remote erben und RemoteException werfen entfällt~~
 - ~~java.io.Serializable aber immer noch erforderlich~~
 - ~~Kein try/catch in Domain Logik~~
- ~~Keine ServiceProvider/ServiceProxy pro fachlichen Service~~
- ~~Keine Classloading Hacks~~
- **Endpoint description ist (zu) statisch (kümmern wir uns gleich drum)**
 - **Code- oder Konfigurationsänderungen, wenn Endpoint wechselt**
 - **Service Provider muss vor dem Service Consumer gestartet sein**

1. Bestandsaufnahme gestrige RMI-Lösung
2. Das Eclipse Communication Framework und „Remote OSGi“
 1. Remote Services
 2. ECF Discovery
 3. *Fallacies of distributed computing*
 4. Provider Architektur
 1. R-OSGi vs Generic
 2. (DNS-SD)
 5. Blackbox-Bundles
 1. R-OSGi Smart Serialization
 2. R-OSGi Smart Proxies
 6. Asynchronous
 1. Callbacks & Futures

- § Implementieren Sie eine ECF-basierte Lösung, die das Transparenz-Problem 4. behebt

- § Vorüberlegung:
 - Muss eine „Endpoint“-Konfiguration (4.) überhaupt auf der „Consumer“-Seite abgelegt werden?
 - Wäre eine zentrale Konfiguration sinnvoller?
 - Single Point of Failure vs. Redundanz
 - Dynamisches Finden (Discovery) von Konfigurationen/Endpoints
 - Aber: „*The fallacies of Distributed Computing*“
 - Setzt bestimmte Topologie voraus

Aufgabe 2 - Das SOA Dreieck



[Champion et al. 2002]

- A protocol and „space“ agnostic API for service discovery
 - Not bound to OSGi
 - Does not expose provider/protocol internals
 - Namespace/ID allows flexibility in service addressing
 - *providerAService.equals(providerBService);*
 - Not limited to, e.g., the local subnet (LAN)
 - However some providers are restricted
 - No guarantees (just because something is discoverable, does not mean it is there)
 - Upper layers may fail to connect
- Provides *IDiscoveryLocator* and *IDiscoveryAdvertiser*
 - Locator finds services
 - Advertiser registers/announces services
 - Consumer gets hold of discovery services
- Transparent when used with RFC 119 (*ServicePublication*)

- **Use case:**
Discovery services
in a (highly)
dynamic network
 - Even without
central (server)
infrastructure
 - Peer2Peer

Dynamic Configuration of IPv4 Link-Local Addr (IPv4LL)

- (link-local/same physical link) 169.254.0.0/16 - RFC 3927

Multicast DNS (mDNS): Peer2Peer name resolution

- Idea: Hosts are authoritative for their resources
- Inherently incompatible with unicast DNS “.local” zones

DNS based Service Discovery (DNS-SD):

- Sits on top of mDNS
- Uses existing DNS SRV and TXT records to compose service descriptions
- Service identity is achieved by instance names (“Markus’ printer, 1st Floor.kuppe.org”)
- Allows delegation for subdomains, like it is possible in “regular” DNS

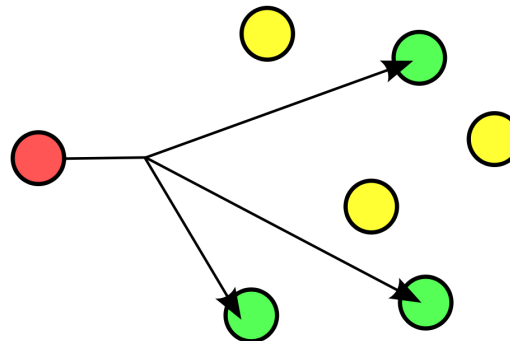
One shot and continuous queries

Well-known as Zeroconf/Apple Bonjour

Aufgabe 2 - IP Multicast - RFC 1112,1458, 2236...

“You put packets in at one end, and the network conspires to deliver them to anyone who asks” [Dave Clark] or “When unicast is door to door, multicast is a radio station with receivers tuning into the right frequency“

- Every IP datagram whose destination address starts with “1110” is a multicast datagram
- Remaining (28) bits identify the multicast “group”
 - To receive multicast message, join a group
- Multicast is handled at the transport layer (OSI layer 4) with UDP
 - TCP provides point-to-point, thus not feasible for multicast
- Default TimeToLive (TTL) of 1 which restricts datagrams to the local subnet, 255 means no restriction at all



Aufgabe 2 - Lessons Learned

- Discovery ermöglicht eine „Service Registry on Steroids“
- Konfiguration entfällt bzw. wird dynamisch
 - Aber: *Fallacies of distributed computing*

1. Bestandsaufnahme gestrige RMI-Lösung
2. Das Eclipse Communication Framework und „Remote OSGi“
 1. Remote Services
 2. ECF Discovery
 3. *Fallacies of distributed computing*
 4. Provider Architektur
 1. R-OSGi vs Generic
 2. (DNS-SD)
 5. Blackbox-Bundles
 1. R-OSGi Smart Serialization
 2. R-OSGi Smart Proxies
 6. Asynchronous
 1. Callbacks & Futures

Zusammenfassung RMI - Was ist aus Sicht von Transparenz störend?

- ~~1. *Serializable, Remote, throws RemoteException* kontaminieren die POJOs mit technischen Details der Remote-Technologie~~
 - ~~• Austausch der Remote-Technologie ist kaum bis gar nicht möglich~~
 - ~~– Lösung: Abstrahieren von *java.rmi.Remote*?~~
 - ~~• Wann ist *throws RemoteException* sinnvoll?~~
- ~~2. *try/catch RemoteException* in Service Nutzern/Domain Logik (Kontaminierung)~~
- ~~3. Ein *ServiceProxy* und *ServiceProvider* Bundle für:~~
 - ~~1. Jeden *Service*~~
 - ~~2. Jede *Remote-Technologie*~~
- ~~4. RMI-Registry muss global bekannt sein~~
 - ~~• Lösung: In XML Konfiguration auslagern?~~
- ~~5. (nicht portable) Classloading Tricks an OSGi vorbei~~
- ~~6. Blockierendes Verhalten synchroner Kommunikation~~

The Fallacies of Distributed Computing

- The network is reliable
 - Latency is zero
 - Bandwidth is infinite
-
- The network is secure
 - Topology does not change
 - There is one administrator
 - Transport cost are zero
 - The network is homogeneous

[Arnon Rotem-Gal-Oz 1994]

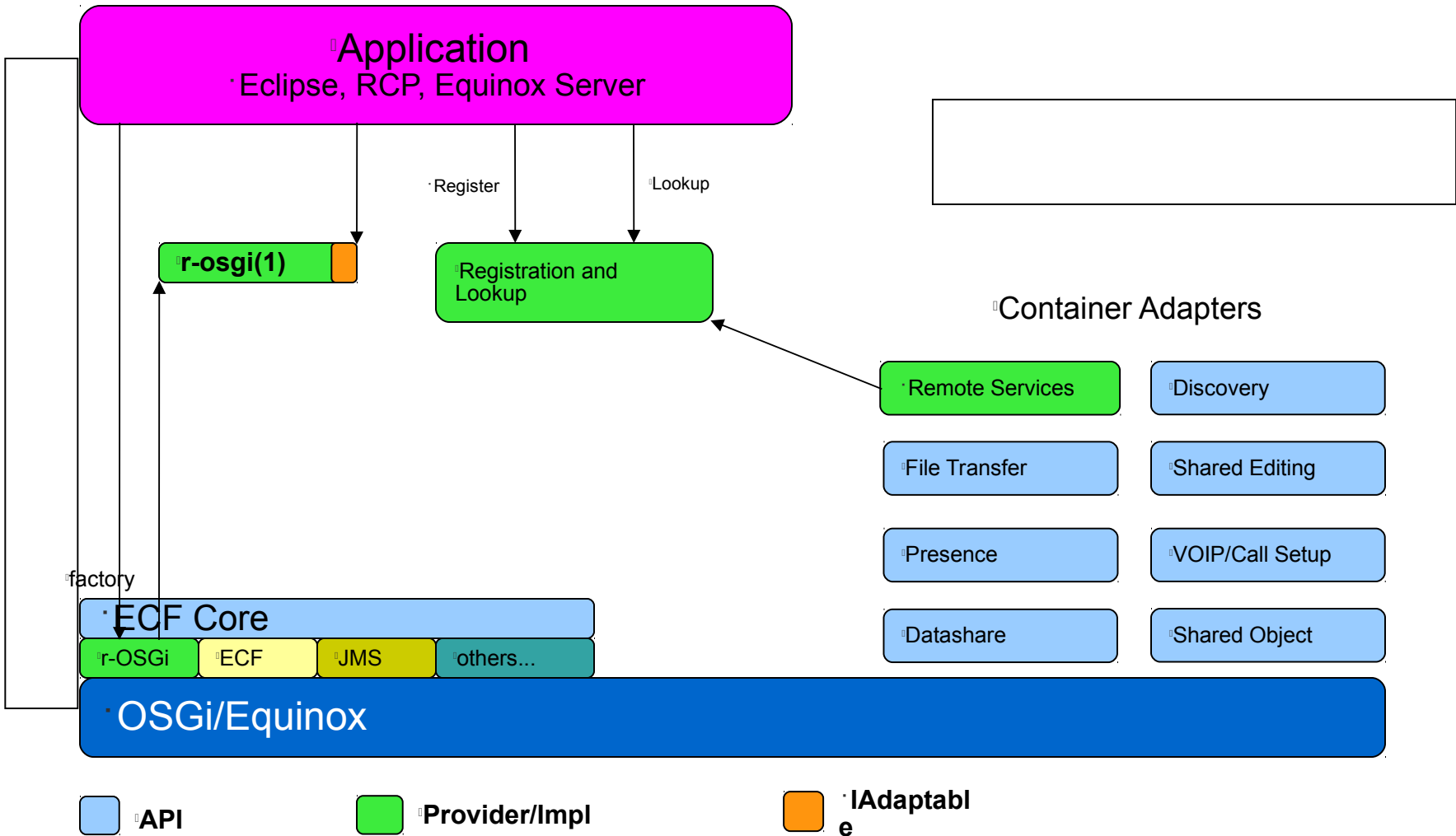
- Erweitern Sie die aktuelle Lösung, so dass dynamisch auf Service-Lifecycle-Events im *ServiceConsumer* reagiert wird
- Vorüberlegung:
 - Welche Lifecycle-Events gibt es denn überhaupt?
 - Services kommen und gehen (das passiert bereits bei lokaler Entwicklung)
 - Das Netzwerk ist nicht zuverlässig

Aufgabe X - ServiceException.REMOTE

```
public void doSave(IProgressMonitor monitor) {
    if (kundenService != null) {
        try {
            kundenService.aktualisiereKunde(currentKunde);
            originalKunde = currentKunde.copy();
            firePropertyChange(EditorPart.PROP_DIRTY);
        } catch (ServiceException e) {
            int type = e.getType();
            switch (type) {
                case ServiceException.REMOTE:
                    // Open dialog informing the user that
                    // a network problem has occured
                    break;
                default:
                    //do something useful in case of local exception
                    break;
            }
        }
    }
}
```

1. Bestandsaufnahme gestrige RMI-Lösung
2. Das Eclipse Communication Framework und „Remote OSGi“
 1. Remote Services
 2. ECF Discovery
 3. *Fallacies of distributed computing*
 4. Provider Architektur
 1. R-OSGi vs Generic
 2. (DNS-SD)
 5. Blackbox-Bundles
 1. R-OSGi Smart Serialization
 2. R-OSGi Smart Proxies
 6. Asynchronous
 1. Callbacks & Futures

ECF Provider Architektur



- Distributed Shared Object (DSO) model
- Proactive, client/server model
 - In case of XMPP transport, the service is hidden
- Connected clients see all service proxies
- By default, no type injection
 - The assumption is that dependant types referenced by the service interface are known to all peers
 - Can be customized to go further
- Can be used with XMPP, JMS, JavaGroup...

- R-OSGi was one of the first projects to enable remote OSGi services
- Is itself „just a service“
- Picks up services tagged for remote access (as does OSGi remote services)
- Protocol and transport independent
- Supports type/bundle injection
 - Client builds a dynamic proxy (bundle)

Aufgabe X - Austausch der ECF Provider

- Ersetzen Sie den ECF Generic (Distribution) Provider mit dem r-OSGi Provider
- Vorüberlegung:
 - Wie teilen wir ECF mit, dass r-OSGi genutzt werden soll?
 - *IHost|ProxyContainerFinder*
 - *-Dorg.eclipse.ecf.osgi.services.distribution.defaultConfigType=ecf.r_osgi.peer*
 - Whiteboard-Pattern mit Service.RANK
 - <https://bugs.eclipse.org/326132>

- **Use case:** File based discovery **on the server side**
😊
 - Wide-area (not bound to subnet borders)
 - Reuses existing infrastructure
 - Centrally managed
 - No additional ports (plain DNS queries TCP/UDP on 53)
 - Very efficient due to caching
 - DNS well established/known
 - Service announcement via DNS Dynamic update (well known?)

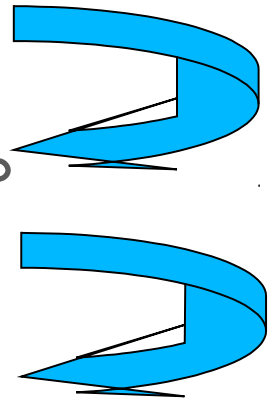
DNS-SD mit Bind Zonefile

```
@ IN SOA ns.smartcity.com.  
    hostmaster.eclipse.org.  
  
(2008110408 18800 18800 604800 86400)  
  
@ IN NS ns.smartcity.com.  
HINFO "i686" "linux 2.6,,
```

...

```
_services._dns-sd._udp IN PTR _http._tcp
```

```
_http._tcp 3600 IN SRV 10 0 80  
    www.eclipse.org.
```



- **Use case:** All (available) discovery providers at once
- While providing the same interface to clients
- Does not filter redundant *IServiceEvents* (yet)
- Dynamic enabled
 - Stores service registrations to reregister with newly added providers

- Die ECF Provider-Architektur ermöglicht es, bestimmte Architektur-Entscheidungen bis zum Deployment-Zeitpunkt zu vertagen
- Unterschiedliche Topologien erfordern unterschiedliche Lösungen
 - Discovery, RemoteServices, Presence, ...
 - OSGi Remote Service Admin Spezifikation sieht eine ähnliche Lösung speziell für RemoteServices vor
 - <https://bugs.eclipse.org/324215>
- Konfiguration ist (teilweise) Provider-abhängig und muss angepasst werden
 - Java System Properties (-Dorg.eclipse.ecf...=12345)
 - *org.osgi.service.cm.ConfigurationAdmin*
 - Programmatisch/deklarativ (bei der Service Registrierung)

1. Bestandsaufnahme gestrige RMI-Lösung
2. Das Eclipse Communication Framework und „Remote OSGi“
 1. Remote Services
 2. ECF Discovery
 3. *Fallacies of distributed computing*
 4. Provider Architektur
 1. R-OSGi vs Generic
 2. (DNS-SD)
 5. Blackbox-Bundles
 1. R-OSGi Smart Serialization
 2. R-OSGi Smart Proxies
 6. Asynchronous
 1. Callbacks & Futures

- Bei Blackbox (Glassbox) Bundles sind Anpassungen für Remoting ausgeschlossen
 - Sourcen stehen nicht zur Verfügung
 - Änderungen sind auf Grund der Lizenz verboten
 - ...
- Welche Änderungen wären nötig?
 - Service und Parameter im Blackbox Bundle implementieren kein *Serializable*
 - Bestimmte Methoden sind für das Remoting ungeeignet
 - *public BigObject doSomething(byte[] aGB)*
 - ...
- Trotzdem müssen wir die Services des Bundles remote zur Verfügung stellen
 - => Lösung: „Smart Serialization“ und „Smart Proxies“ in r-OSGi

Aufgabe X - Smart Serialization

- Betrachten Sie alle Bundles aus dem de.c1wps Namensraum als Blackbox Bundles und entfernen Sie daher „*implements Serializable*“ aus den Service Interfaces und Parameter-Klassen

- Anforderung: Die Methode *de.c1wps.winterschool.domain.kunde.service.IKundenService.aktualisiereKunde(Kunde)* soll vor auf der ServiceConsumer-Seite prüfen, ob der Kunde gültig ist
- Aufgabe: Implementieren Sie einen r-OSGi SmartProxy, der den Aufruf *IKundenService.aktualisiereKunde(Kunde)* auf der ServiceConsumer-Seite abfängt (intercept) und prüfen Sie die Kundennummer auf eine *NumberFormatException*
 - Diese Prüfung ist (natürlich) redundant zu *Kundennummer*

Aufgabe X - SmartProxies mit r-OSGi ECF remoteservice Provider

```
package de.clwps.winterschool.service.kunde;

import de.clwps.winterschool.domain.kunde.material.Kunde;
import de.clwps.winterschool.domain.kunde.service.IKundenService;

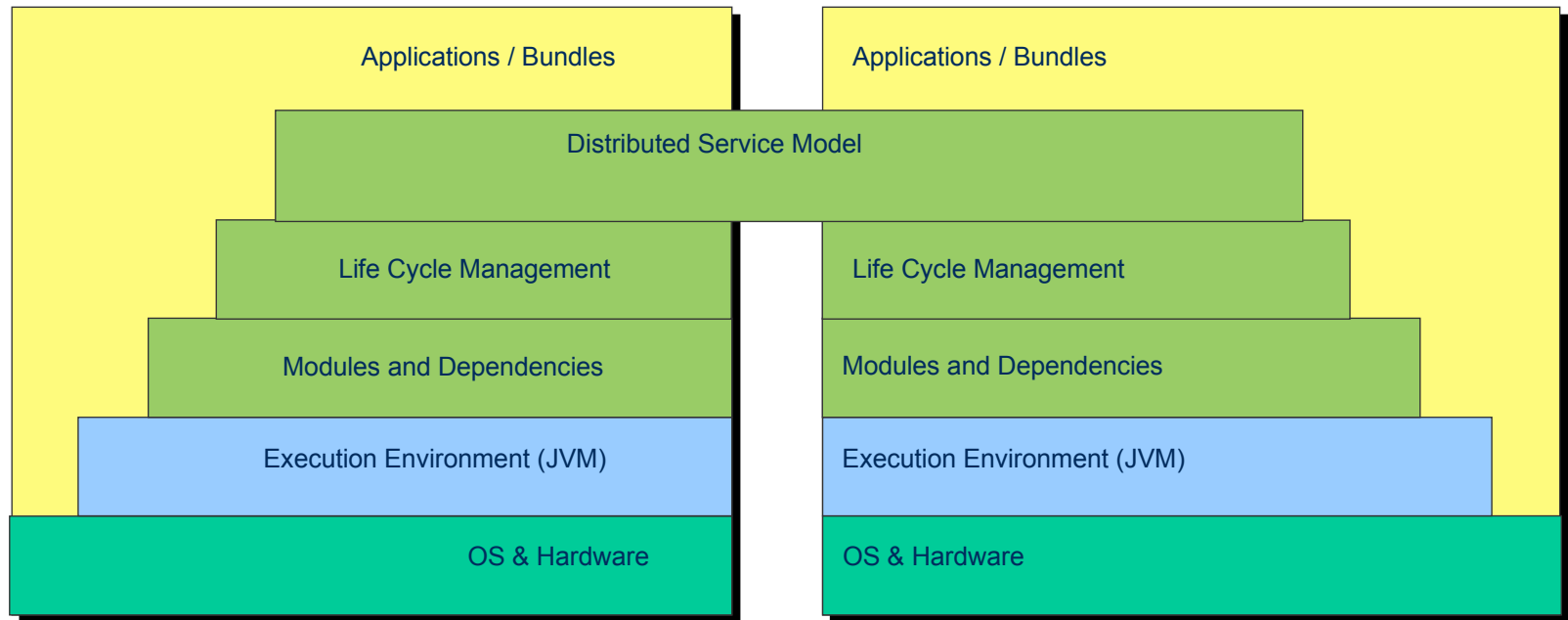
public abstract class SmartKundenServiceMemImpl implements IKundenService {

    public abstract boolean _rosgiAktualisiereKunde(Kunde kunde);

    /* (non-Javadoc)
     * @see
     * de.clwps.winterschool.domain.kunde.service.IKundenService#aktualisiereKunde(
     * de.clwps.winterschool.domain.kunde.material.Kunde)
     */
    public boolean aktualisiereKunde(Kunde kunde) {
        System.out.println("smartproxy");
        return _rosgiAktualisiereKunde(kunde);
    }
}
```

- SmartProxy muss Original-Interface implementieren
 - Original-Methoden stehen über abstrakte Deklaration mit Präfix „_rosgi“ zur Verfügung
- Registrierung über `ch.ethz.iks.r_osgi.RemoteOSGiService.SMART_PROXY`
 - Abhängigkeit auf r-OSGi

Blackbox Bundles mit r-OSGi



Aufgabe X - Lessons Learned

- SmartSerialization erlaubt Blackbox Bundles
 - (*native* Code scheitert)
 - Aber aufpassen, was die SmartSerialization macht (BigObject)
- SmartProxies ermöglichen das dynamische Intercepted/Dispatchen von Methodenaufrufen
 - Aufwändige Berechnungen können auf die ServiceConsumer-Seite ausgelagert werden
 - Extra Prüfungen können bereits im Consumer durchgeführt werden
- Definition des SmartProxys erfolgt auf der ServiceProvider-Seite
 - Nur Möglich, da r-OSGi auch auf dem Module-Layer arbeitet (OSGi remote services nicht!)
 - SmartProxies erfordern keine Änderung auf Seiten des ServiceConsumers
- Invasive Methode vs. Surrogate (<https://bugs.eclipse.org/327063>)
- Aber: Smart Proxies sind nicht standardisiert (nur r-OSGi Provider)
 - Standardisierung wäre technische möglich

1. Bestandsaufnahme gestrige RMI-Lösung
2. Das Eclipse Communication Framework und „Remote OSGi“
 1. Remote Services
 2. ECF Discovery
 3. *Fallacies of distributed computing*
 4. Provider Architektur
 1. R-OSGi vs Generic
 2. (DNS-SD)
 5. Blackbox-Bundles
 1. R-OSGi Smart Serialization
 2. R-OSGi Smart Proxies
 6. Asynchronous
 1. Callbacks & Futures

Zusammenfassung RMI - Was ist aus Sicht von Transparenz störend?

- ~~1. *Serializable*, *Remote*, *throws RemoteException* kontaminieren die POJOs mit technischen Details der Remote-Technologie~~
 - ~~• Austausch der Remote-Technologie ist kaum bis gar nicht möglich~~
 - ~~- Lösung: Abstrahieren von *java.rmi.Remote*?~~
 - ~~• Wann ist *throws RemoteException* sinnvoll?~~
- ~~2. *try/catch RemoteException* in Service Nutzern/Domain Logik (Kontaminierung)~~
- ~~3. *ServiceProxy* und *ServiceProvider* Bundle für jeden Service~~
- ~~4. RMI-Registry muss global bekannt sein~~
 - ~~• Lösung: In XML Konfiguration auslagern?~~
- ~~5. (nicht portable) Classloading Tricks an OSGi vorbei~~
- 6. Blockierendes Verhalten synchrone Kommunikation**

§ Synchroner Kommunikation



§ Asynchrone Kommunikation



Aufgabe X - Asynchrone Kommunikation

- § Implementieren Sie eine ECF-basierte Lösung, die das Transparenz-Problem 6. behebt

- § Vorüberlegung:
 - Geht das auch mit Hausmitteln?!
 -

- Service Interface Name ist [Originaler Name]Async
- Interface muss von *org.eclipse.ecf.remoteservice.IAsyncRemoteServiceProxy* erben
 - Bundle-Abhängigkeit zu ECF
- Jede asynchrone Methode muss im Interface mit [Originaler Name]Async vorhanden sein
 - Zusätzlicher Parameter *IAsyncCallback<Typ des originalen Rückgabewerts>*
 - Rückgabewert ist *void*

Aufgabe X - Asynchrone Kommunikation

```
package de.clwps.winterschool.domain.kunde.service;

import org.eclipse.ecf.remoteservice.IAsyncCallback;
import org.eclipse.ecf.remoteservice.IAsyncRemoteServiceProxy;

import de.clwps.winterschool.domain.kunde.fachwerte.Kundennummer;
import de.clwps.winterschool.domain.kunde.material.Adresse;
import de.clwps.winterschool.domain.kunde.material.Kunde;

public interface IKundenServiceAsync extends IAsyncRemoteServiceProxy {

    void listKundenAsync(IAsyncCallback<Kunde[]> callback);

    void existiertKundeAsync(Kundennummer kundennummer, IAsyncCallback<Kundennummer> callback);

    void erzeugeKundeAsync(String vorname, String nachname, Adresse adresse, IAsyncCallback<Object> callback);

    void aktualisiereKundeAsync(Kunde kunde, IAsyncCallback<Boolean> callback);

    void getKundeAsync(Kundennummer kundennummer, IAsyncCallback<Kunde> callback);

    void naechsteKundennummerAsync(IAsyncCallback<Kundennummer> callback);

    void clearAsync(IAsyncCallback<String> callback);

}
```

- Nachteile: Originales Interface und *Async stelle eine Art Fork dar
 - Lösung: Async Annotations (<https://bugs.eclipse.org/309732>)

Aufgabe X - Asynchrone Kommunikation

```
public void doSave(IProgressMonitor monitor) {
    if (kundenService != null) {
        if (kundenService instanceof IKundenServiceAsync) {
            IKundenServiceAsync kundenServiceAsync =
                (IKundenServiceAsync) kundenService;
            kundenServiceAsync.aktualisiereKundeAsync(
                currentKunde, new IAsyncCallback<Boolean>() {

                public void onSuccess(Boolean arg0) {
                    System.out.println("Result is: " + arg0);
                }

                public void onFailure(Throwable arg0) {
                    arg0.printStackTrace();
                }

            });
        } else {
            kundenService.aktualisiereKunde(currentKunde);
        }
        originalKunde = currentKunde.copy();
        firePropertyChange(EditorPart.PROP_DIRTY);
    }
}
```

Aufgabe X - Asynchrone Kommunikation mit IFuture

```
public void doSave(IProgressMonitor monitor) {
    if (kundenService != null) {
        if (kundenService instanceof IKundenServiceAsync) {
            IKundenServiceAsync kundenServiceAsync =
                (IKundenServiceAsync) kundenService;
            IFuture future =
                kundenServiceAsync.aktualisiereKundeAsync(currentKunde);
            future.cancel(); // Cancel aktualisiereKundeAsync(...)
        } else {
            kundenService.aktualisiereKunde(currentKunde);
        }
        originalKunde = currentKunde.copy();
        firePropertyChange(EditorPart.PROP_DIRTY);
    }
}
```

- ~~Hier org.eclipse.equinox.concurrent.future.IFuture~~
- In Zukunft wird `java.util.concurrent.future` dieses Programmiermodell weiter bekannt machen
 - Java nutzt „explizite“ Future, da `future.get()` aufgerufen werden muss
 - Andere Sprachen nutzen implizite Future (dort allerdings in die Sprache eingebaut)
 - Vgl. `doesNotUnderstand` in Smalltalk

- Asynchrone Kommunikation erfordert ein neues Programmiermodell
 - Teilweise bekannt aus UI-Entwicklung (Listener Ansatz)
 - Features in Java bisher neu, andere Sprachen setzen dieses Modell stärker ein
- Abhängigkeiten auf *org.eclipse.ecf.remoteservice* entstehen im Code
 - *org.eclipse.equinox.concurrent.future.IFuture*
- *IAsyncRemoteService* ist kein OSGi Standard
 - Aber Arbeiten/Diskussionen um Asynchronität innerhalb von OSGi haben begonnen



Danke für die Aufmerksamkeit!

Fragen/Anmerkungen?

- [Champion et al 2002] - Miachel Champion, Chris Ferris, Eric Newcomer and David Orchard: „*Web Services Architecture*“, <http://www.w3.org/TR/2002/WD-ws-arch-20021114/> , [Online; Stand 20. Dezember 2009]
- [Arnon Rotem-Gal-Oz 1994] - Arnon Rotem-Gal-Oz: „*Fallacies of Distributed Computing Explained*“ <http://www.rgoarchitects.com/Files/fallacies.pdf> , , [Online; Stand 1. Oktober 2010]
- Numerous ECF committers

- IBM and the IBM logo are trademarks or registered trademarks of IBM Corporation, in the United States, other countries or both.
- Java and all Java-based marks, among others, are trademarks or registered trademarks of Sun Microsystems in the United States, other countries or both.
- OSGi is a trademark or registered trademark of the OSGi Alliance in the United States and other countries.
- Eclipse and the Eclipse logo are trademarks of Eclipse Foundation, Inc.
- Other company, product and service names may be trademarks or service marks of others.