Tutorial: Distributed OSGi – The ECF way

Markus Alexander Kuppe
Scott Lewis

http://www.eclipse.org/ecf

Distributed OSGi – The ECF way

- This **_is_** a tutorial on
  - **ECF remoting/distribution API + impls**
  - **RFC 119 (EE spec for Distributed OSGi)**
  - **ECF discovery API + impls**
- This is **_not_** a tutorial on
  - OSGi Services
  - Equinox
  - Declarative Services
  - Apache CXF
  - Extension registry and/vs. OSGi Services
- **This tutorial is arranged as a set of labs/modules**

Tutorial schedule - Agenda

13:30 – 13:45	**Welcome + Introduction + Teams**
13:45 – 14:15	**Lab 1: *Non-Transparent remote services with ECF***
14:15 – 14:30	**Wrap-up lab1 (Remote service best pratice)**
14:30 – 15:00	**Lab 2: *Transparent remoting with RFC 119***
15:00 – 15:15	**Wrap-up lab2 (Changes in OSGi 4.2)**
15:15 – 15:45	**Break**
15:15 – 16:15	**Lab3: *Adding service discovery via pluggable ECF discovery and SLP and mDNS***
16:15 – 16:30	**Wrap-up lab3 (Multicast based discovery providers)**
16:30 – 17:00	**Lab 4: *Implement a wide area discovery provider based on DNS-SD***
17:00 – 17:30	**Wrap up and *Distributed Testing with ECF* plenum**

Labs

- Lab1
  - org.eclipse.ecf.examples.remoteservices - Create a “server” and “client” bundle and make it work with non-transparent ECF remoting
- Lab2
  - Turn the non-transparent Lab1 into transparent remoting with RFC119 (get rid of boiler-plate code)
  - Talk about transparency, remote interfaces/contracts and ceavats (e.g. what about errors, blocking, etc)
- Lab3
  - Add service discovery to get rid of static configuration through the service file
  - Explore discovery protocols and their limitations + point out multicast obstacles
- Lab4
  - Implement your own ECF wide-area discovery provider based on DNS-SD and learn how to set up zone files (Alternativley use UDDI, JINI, XMPP ServiceDiscovery,…)

If there is still time left…

- Lab5
  - Use traffic sniffer and inspect what is going on on the wire → be alarmed about no security whatsoever
- Lab6
  - Implement a secure network channel for r-OSGi (http://r-osgi.sourceforge.net/channels.html)
- Lab7
  - Extend ECF discovery UI with a custom EMF model for a specific service

What you should learn/Goals for today

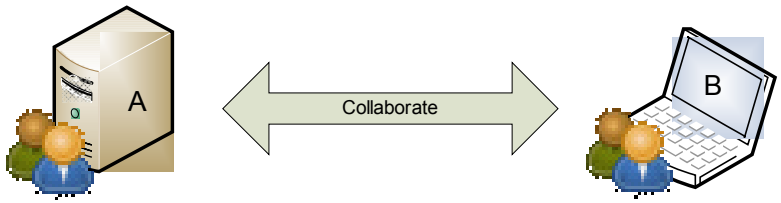
- Setup/use ECF distribution/remoting with one/several providers
  - Implement remote service **consumer**
    - Transparent Proxy
    - Other pattern (e.g. async with listener, future)
  - Implement remote service **provider**
  - Testing/Debugging
  - Selecting one/several providers
- Setup/use ECF discovery with one/several providers
  - Discover existing/published services (e.g. iTunes, others)
  - Publish (and discover) own services
  - Debug discovery
    - Wireshark
    - Discovery UI
  - Selecting one/several providers

Team set up

- This is a double diamond talk. See us (Scott and Markus) as „facilitators of ideas“ or coachs, definitely not lecturer!
  - We don‘t have all the answers anyway :-)

- We always welcome feedback
  - Speak up if you have something to say
  - We want the teams (you) to wrap up labs (your lesson learned)

- We intent to run this tutorial in teams :-)
  - Please build groups of ~four
  - Team A will work on the server, team B on the client impl
  - In case you‘d rather like to work alone, **that is fine too**



What you need

- Your laptop
- A working wifi connection
- Eclipse SDK >= 3.5M6
- JavaSE >= 1.4 (preferrably a SUN JDK)
- Traffic Sniffer (e.g. Wireshark - http://www.wireshark.org/)

- Optional: Any JavaSE capable gadget (e.g. mobile phone).
  - We have an iPhone that runs ECF and can be accessed during the tutorial.

What you need cont.

For the impatient…

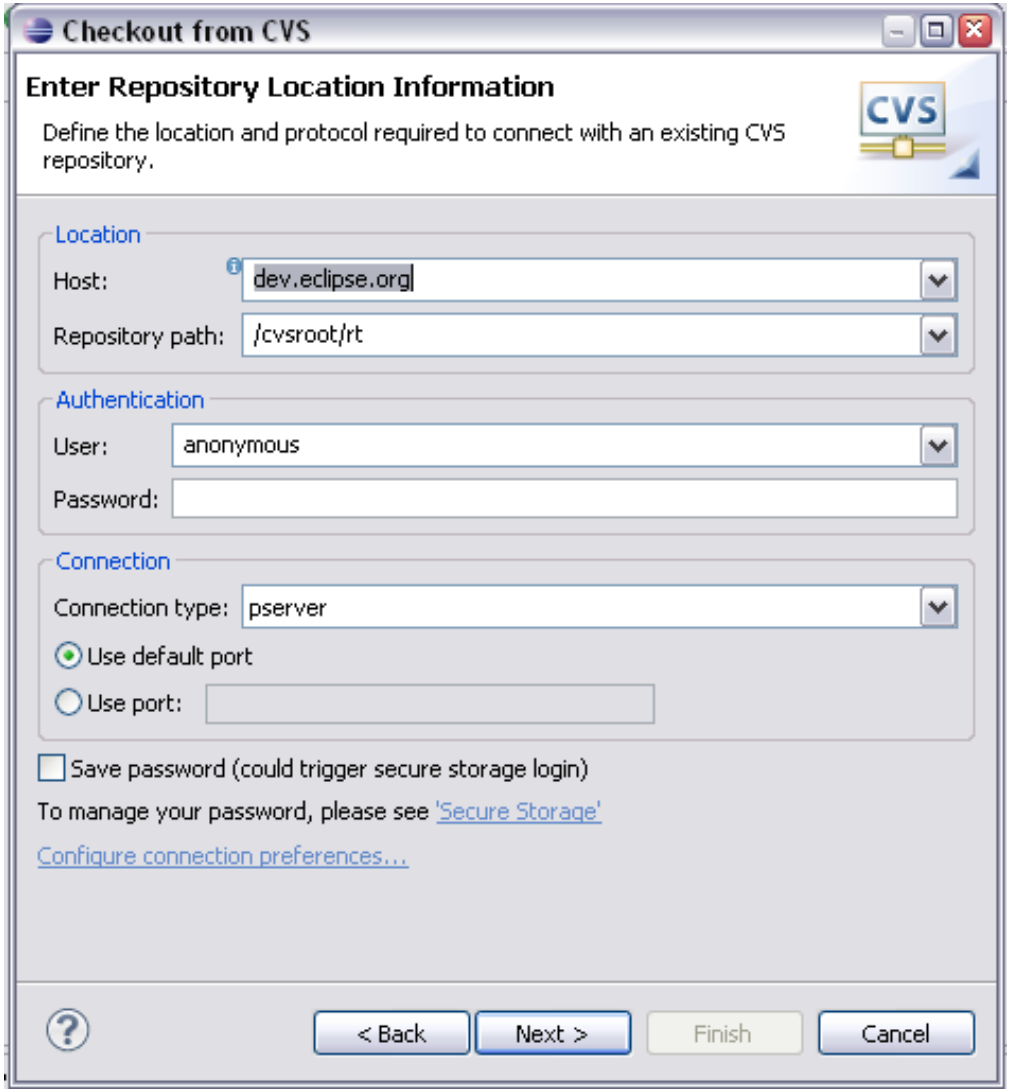
:pserver:anonymous@dev.eclipse.org:/cvsroot/rt/

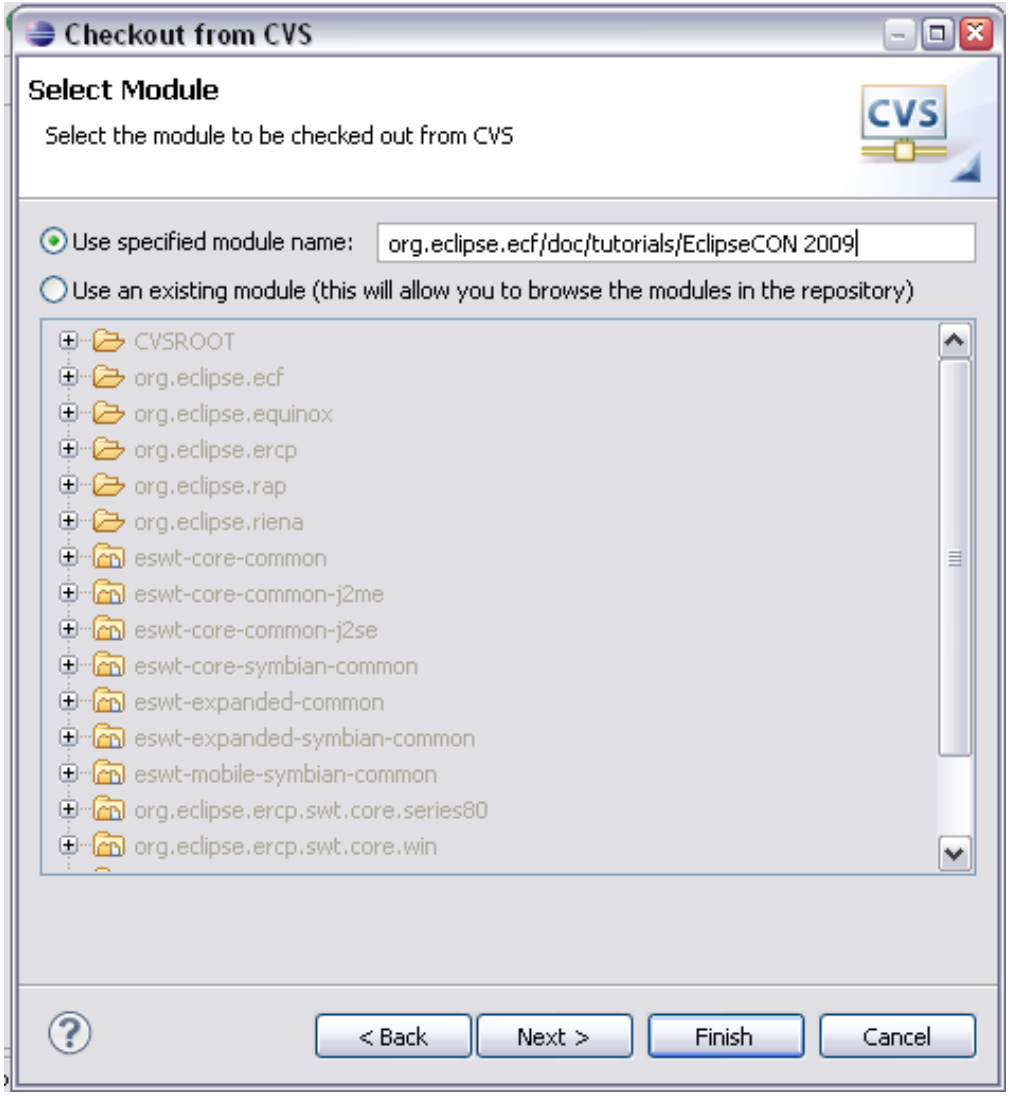
org.eclipse.ecf/doc/tutorials/EclipseCON 2009

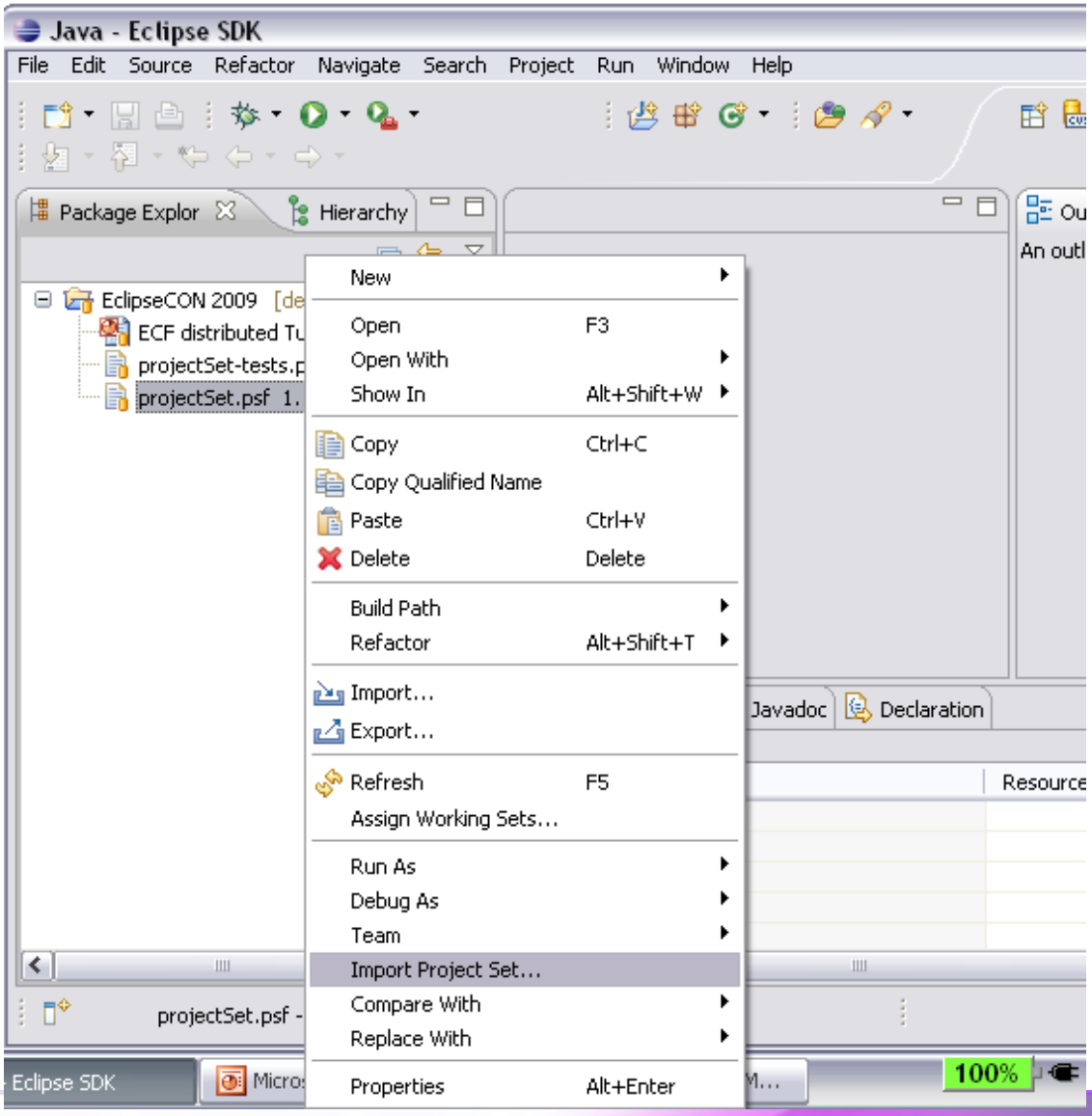
or

CVS FAQ - How do I get a project into my workspace from CVS?

and import projectSet.psf







History of ECF

- 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

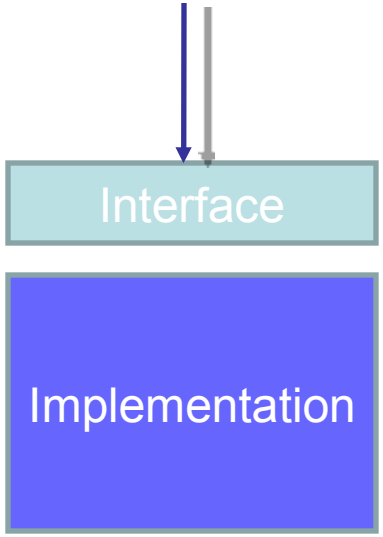
Who is using ECF today?

- 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/

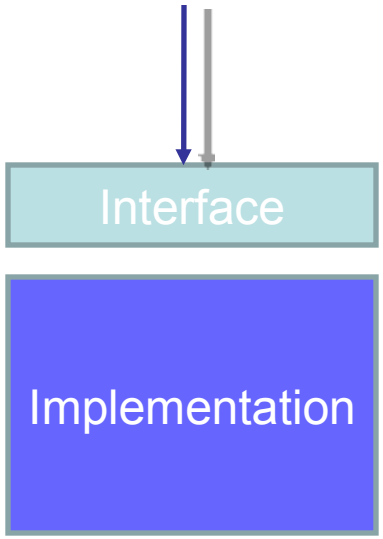
OSGi Services

OSGi Services

- OSGi services provide
  - Encapsulation at a larger granularity
  - Loose coupling of functionality
  - Extensibility and Abstraction

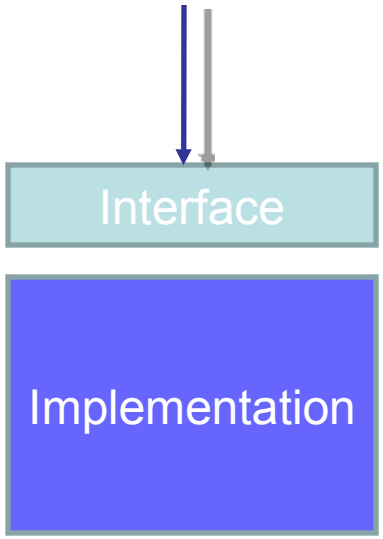


How are OSGi Services Exposed and Used?



- Registration - “service host”
  - BundleContext.registerService(...)
- Lookup/Finding - “service consumer”
  - BundleContext.getServiceReferences(...)
  - BundleContext.getService(ref)
- Use (consumer)
  - Calling methods on interface
    - Implementing object's code is run (duh)
- Clean-up
  - Releasing References (gc for dynamic services)
    - ServiceRegsitration.unregister, BC unget(reference)

But there are multiple methods...even for local OSGi services

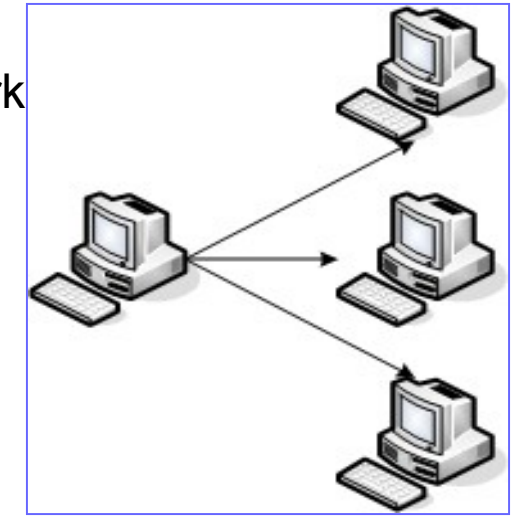


- Registration
  - Declarative Services (DS)
- Lookup
  - ServiceTracker and DS
- Clean-up
  - Releasing References
    - ServiceTracker, DS

- **Why multiple methods?**
  - My Answer: To address various use cases for 'services'
  - Sometimes necessary to manage registration, lookup, cleanup more directly

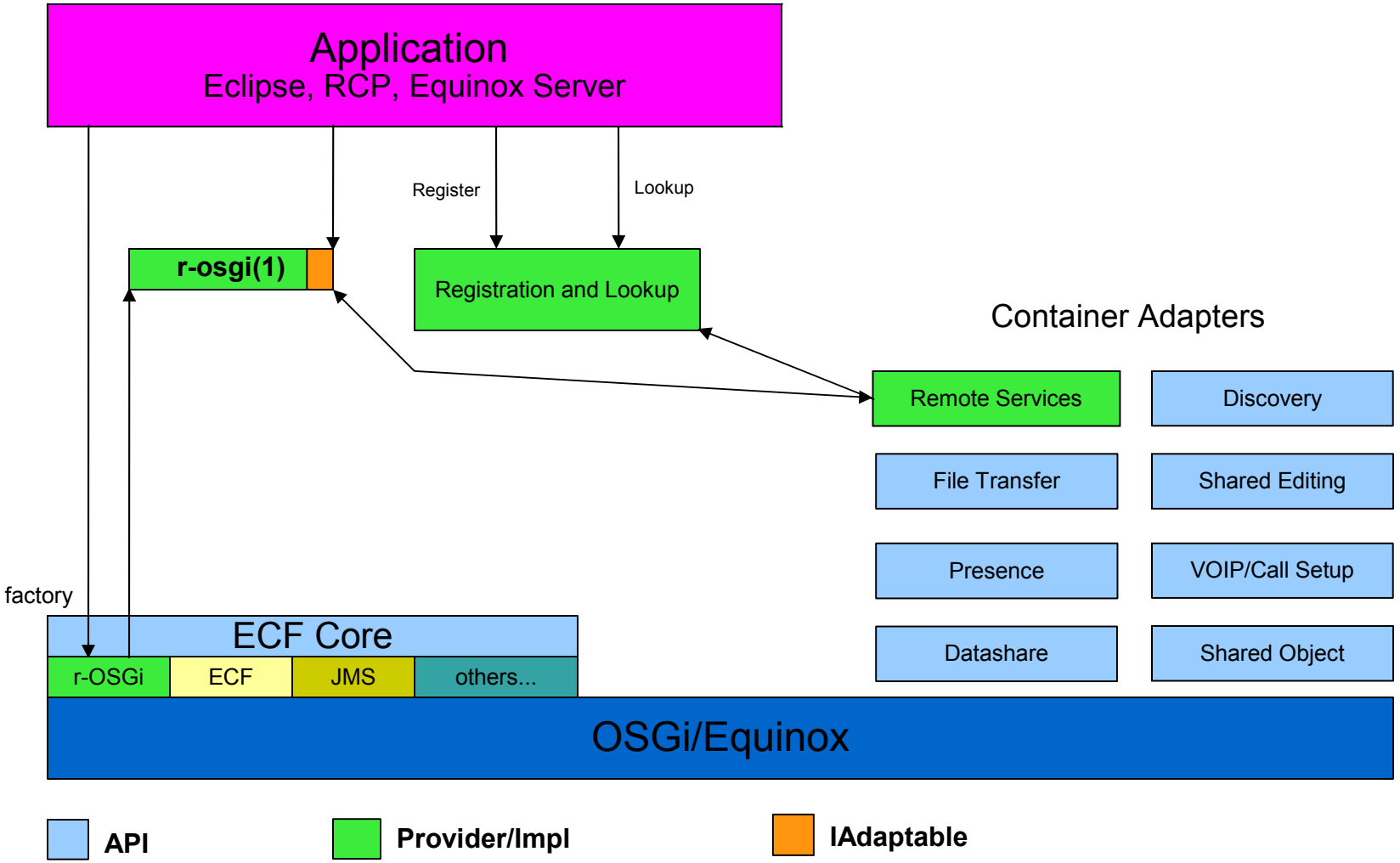
OSGi services in the network

- Registration
  - Extra step: publication for discovery
  - Security implications - now anyone on network can access
- Lookup
  - Network discovery necessary
    - No BC for remote fwk
  - Addressing: Have to identify remote framework out of many
- Cleanup
  - Now to guarantee cleanup with unreliable network



ECF Architecture

ECF Provider Architecture



ECF Remote Services

- API Resembles OSGi (but distinct from)
  - Registration: adapter.registerRemoteService(...)
  - Lookup: IRemoteServiceReference ref = adapter.getRemoteServiceReferences(...)
  - Cleanup: unregister/unget
- We have **used** ECF RS (and Discovery) to implement RFC 119
  - ...so...programmer can use transparent registration/lookup
  - Or non-transparent registration/lookup
  - Have supported intents in ECF core API (all ECF providers can expose/use intents at runtime)
  - We are enabling both
  - Allows distribution providers to reuse standards-compliant impl of RFC 119...i.e. They don't have to track/implement RFC 119...we'll do that for them
- Labs will include transparent and non-transparent registration, lookup, and cleanup

ECF Remote Services Providers

- R-OSGi
- ECF Generic
- XMPP
- JMS
- Skype
- JavaGroups
- Would like others...e.g. Riena, SOAP-based, REST-based, commercial/closed, CXF, etc
- RS API open/open source as are all our implementations
    - API can be extended/enhanced over time
    - Existing provider implementations can be reused
    - e.g. XMPP, JMS, Skype, JavaGroups all reuse ECF Generic
- Any/all ECF RS providers get ongoing, **free** support for RFC 119

R-OSGi Provider

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
Only the interface is transmitted
Client builds a dynamic proxy
Can be added to any OSGi runtime (R3 + R4)
Protocol and transport-independent

Generic Provider

Distributed Shared Object (DSO) model
Proactive, client/server model

In case of XMPP transport, the server is “hidden”

Connected clients see all service proxies
By default, no type injection

the assumption is that dependant types referenced by the service interfaces are known to all peers

Can be customized to go further

Can be used with XMPP, JMS, JavaGroups

Code: Service Host

```
// Create container
IContainer container =
   containerManager.getContainerFactory().createContainer("ecf.r_osgi.pe
   er");

// Get remote services adapter
IRemoteServicesContainerAdapter adapter =
   (IRemoteServicesContainerAdapter)
   container.getAdapter(IRemoteServicesContainerAdapter.class);

// Register IMyService
IRemoteServiceRegistration registration =
   adapter.registerRemoteService(new String[]
   {IMyService.class.getName()}, serviceImplementation,null);


// use registration to manage service
```

Code: Service Consumer

```
// Create container
IContainer container =
    containerManager.getContainerFactory().createContainer("ecf.r_osgi.pe
    er");

// Get remote services adapter
IRemoteServicesContainerAdapter adapter =
    (IRemoteServicesContainerAdapter)
    container.getAdapter(IRemoteServicesContainerAdapter.class);

// Lookup IMyService proxy
IRemoteServiceReference [] references =
    adapter.getRemoteServiceReferences(targetID,IMyService.class.getName(
    ),null);
IRemoteService remoteService = adapter.getRemoteService(references[0]);
IMyService svc = (IMyService) remoteService.getProxy();


// Use svc
```

Lab 1: Implement Service Consumer

- Bundle: org.eclipse.ecf.tutorial.lab1
- Lab1Action - Top level menu/toolbar entry to start with
- Tasks
  - Implement calling proxy to get OS info about remote environment
  - Select one of servers running (on blackboard)
  - Use proxy, async listener, future
  - Multiple server providers: r-osgi and ecftcp
  - Extra credit – do lookup in non-ui thread

Lab 1b: Implement Service Host

- org.eclipse.ecf.examples.remoteservices.common (interface)
- org.eclipse.ecf.examples.remoteservices.server
- (implementation)
- Register with IRemoteServiceContainerAdapter.registerRemoteService
- Publish via IDiscoveryAdvertiser.registerService

RFC 119

- Discovery and Distribution
- Transparent Registration, Lookup, and Cleanup
  - Just BC.registerService and BC.getServiceReferences
  - Allows easy remoting of existing services
  - Don't have to be concerned with details of publication/discovery, network lookup, proxy creation, etc

Not **completely** transparent, as has two new service properties

- Registration: **osgi.remote.interfaces**
- Indicates to distribution provider that it should publish/remote the service
- Consumer: **osgi.remote**
- Indicates to consumer that service is remote

- Intents
  - 'Hints' that allow service hosts to require certain distribution characteristics
    - e.g. reliability, security, passbyvalue/ref, etc

RFC 119 – Service Properties

- Service Host (Registration)
  - Required:
    - **osgi.remote.interfaces** - String[]
  - Optional
    - **service.intents** – String[]
    - **osgi.remote.requires.intents** – String[]
    - **osgi.remote.configuration.type** – String[]
- Use ECFServiceConstants
  - Consult inline documentation for details

- Service Consumer
  - **osgi.remote – Present/set but no RFC119-specified value**

ECF value is IRemoteService instance

  - Allows consumers the flexibility to use alternative calling styles available on IRemoteService
    - One-Way – fireAsync
    - Futures – callAsync/1
    - Async with Listener - callAsync/2

Lab 2a: Service Consumer with RFC 119

Lab 2b: Service Host with RFC 119

Do we want Transparent **Usage**?

(as opposed to registration and lookup?)

- Some unavoidable differences between local and RPC
  - Performance
  - Reliability
  - Marshalling
- Can't Fix Network/Can't Ignore/Hide Network...so what to do?
  - Note on Distributed Computing
  - Best Practices for Distributed OSGi Services (#633)
  - OSGi's Service Model is Dynamic
    - Consumers are supposed to deal with that
  - This aspect of 'network transparency' is controversial
  - We want to allow/support programmers in using both transparent and non-transparent usage

Transparent **Usage** (cont)

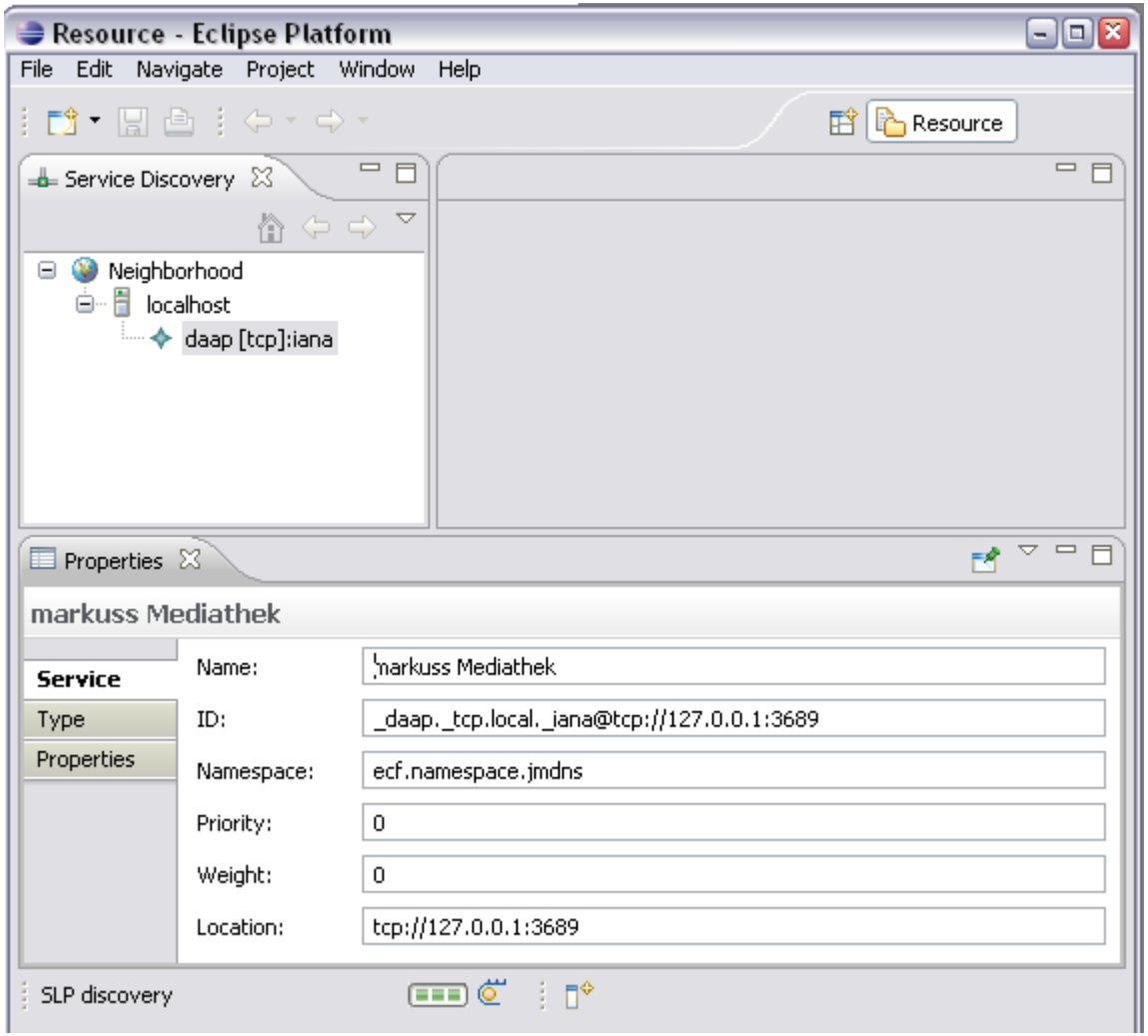
- Transparent Usage Approach
  - Consumers have Proxy only
  - Runtime/unchecked exceptions for network failures
  - Caller/consumer have to deal with blocking/performance
    - This can be **very** problematic
  - RFC 119: Intents allow mechanism for constraining behavior
- ECF
  - Exposes IRemoteService
  - Gives proxy AND additional calling patterns to service consumer
  - AsyncExec, Future, One-Way
- RemoteServiceTracker
  - IRemoteService getRemoteService() rather than Object getService()

ECF Discovery

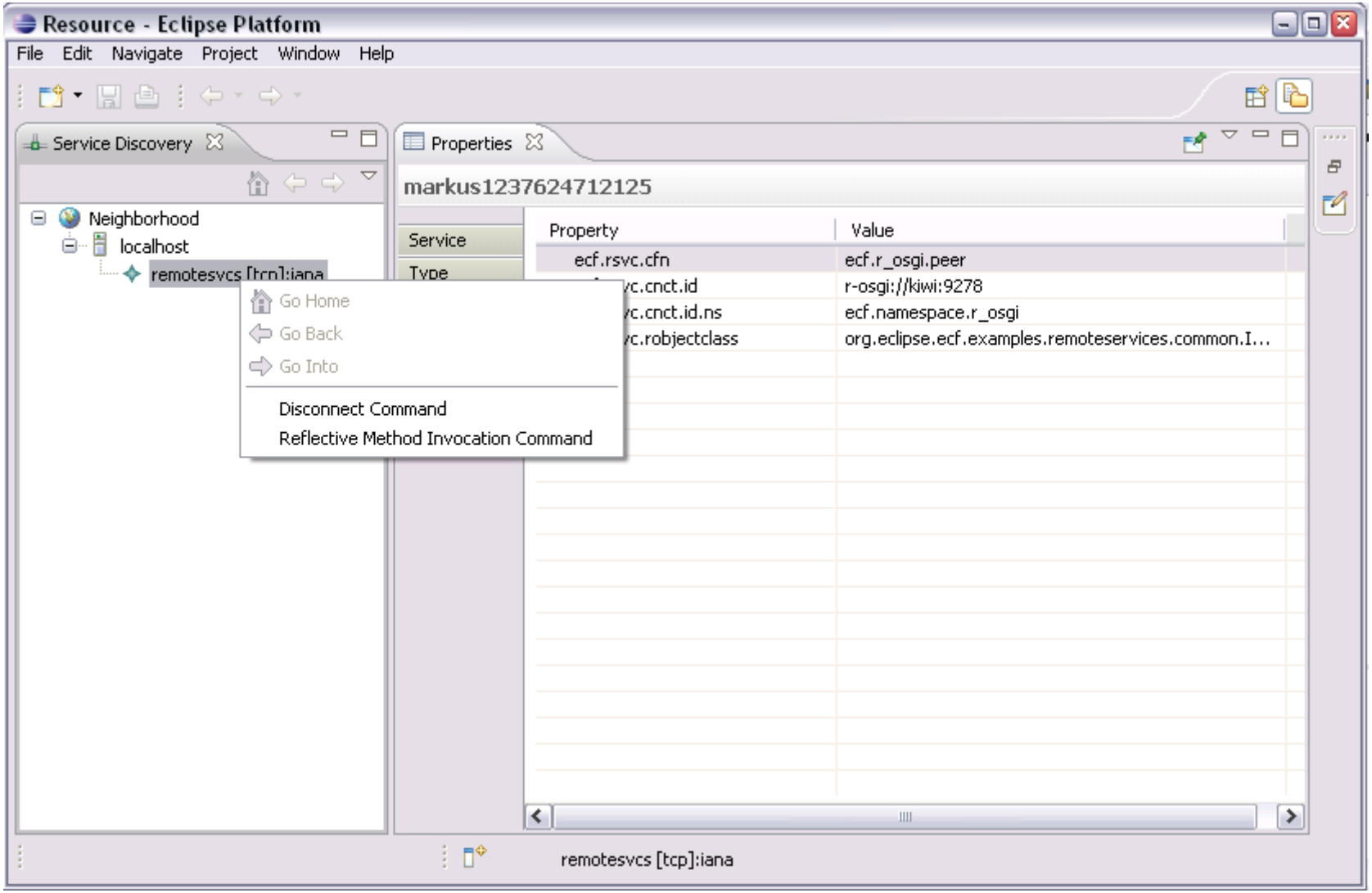
TODO

- Shortcomings ECF discovery API or protocols incompatibilities
- DiscoveryView
  - Show how to start via provided .launch
  - Screenshots with more devices
- Lab4
  - More information on DNS-SD
  - Fix implementation + Unit tests
- Lab5
  - Own example.remoteservices.MyService via UI + EMF model?
- No RFC119 intent definion yet
- Distributed service registry
  - No atomicity (transactions)
  - Just because something is discoverable doesn't mean it is actually there

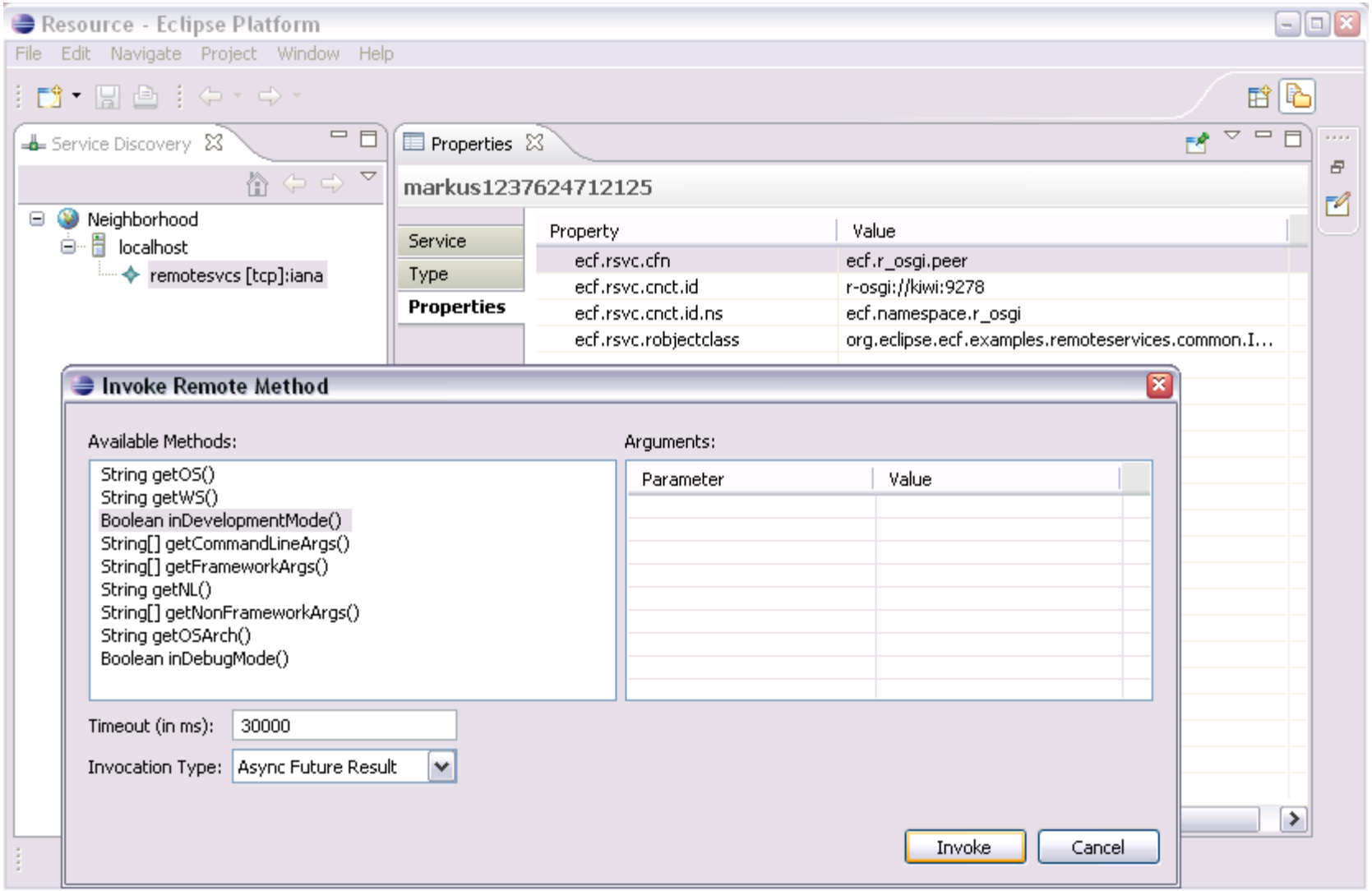
ECF Discovery & Remoteservice UI



ECF Discovery & Remoteservices UI



ECF Discovery & Remoteservices UI



ECF Discovery API – 1k feet above

- 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(provdierBService);`
  - Not limited to, e.g., the local subnet (LAN)
    - However some providers are restricted
  - No garantuees (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`)

Main Interfaces of ECF Discovery

```
// Discovery and register services with…
org.eclipse.ecf.discovery.IDiscoveryLocator
org.eclipse.ecf.discovery.IDiscoveryAdvertiser

// Uniqueness/Identity for service is handled by IDs
org.eclipse.ecf.discovery.identity.IServiceID
org.eclipse.ecf.discovery.identity.IServiceTypeID
// Factory to create new IServiceTypeIDs
org.eclipse.ecf.discovery.identity.IServiceIDFactory

// The actual service instace (used in query by
   example too)
org.eclipse.ecf.discovery.IServiceInfo
Org.eclipse.ecf.discovery.IServiceProperties
```

IDiscoveryLocator|Advertiser

Provides three methods of usage

- Synchronous ***getServices()***, ***getServiceTypes()***,... ***registerServices()*** which block until operation terminates

```
IServiceInfo[] services =
discoveryLocator.getServices();
for (int i=0; i < services.length; i++){
 // do something with the service
```

IDiscoveryLocator|Advertiser

Provides three methods of usage

- Via ***Iservice[Type]Listener*** which will be notified upon a ***IServiceEvent***

```
discoveryLocator.addServiceListener(
 new IServiceListener(){
    public void serviceDiscovered(IServiceEvent
 anEvent) {
 // do something with the service
```

IDiscoveryLocator|Advertiser

Provides three methods of usage

- With *o.e.equinox.concurrent.future.IFuture* (ECF 3.0)

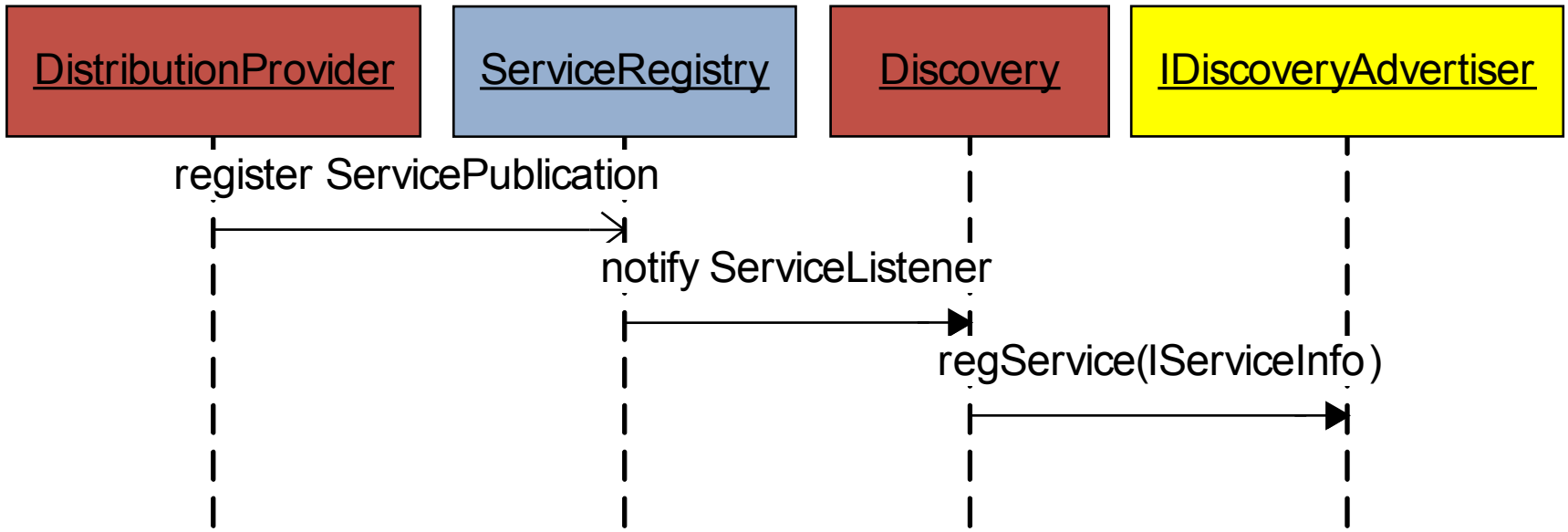
```
IFuture aFuture =
discoveryLocator.getAsyncServices();
// do sth else and let discovery do its job
IServiceInfo[] services = (IServiceInfo[])
aFuture.get();
for (int i = 0; i < services.length; i++) {
 // do something with the service
```

IServiceTypeID & IServiceID & IServiceInfo

```
// First we need a namespace for which the IDs are valid
Namespace namespace =
   discoveryLocator.getServicesNamespace();

// Create a service type identifier
IServiceTypeID serviceTypeID =
   ServiceIDFactory.getDefault().createServiceTypeID(nam
   espace, "http");

// Create a service based on the type (IServiceID will
   be created by the ServiceInfo
URI uri = new
   URI("http://locahost:8080/servlets/myservlet");
IServiceInfo service = new ServiceInfo(uri, "My
   servlet", serviceTypeID);
```

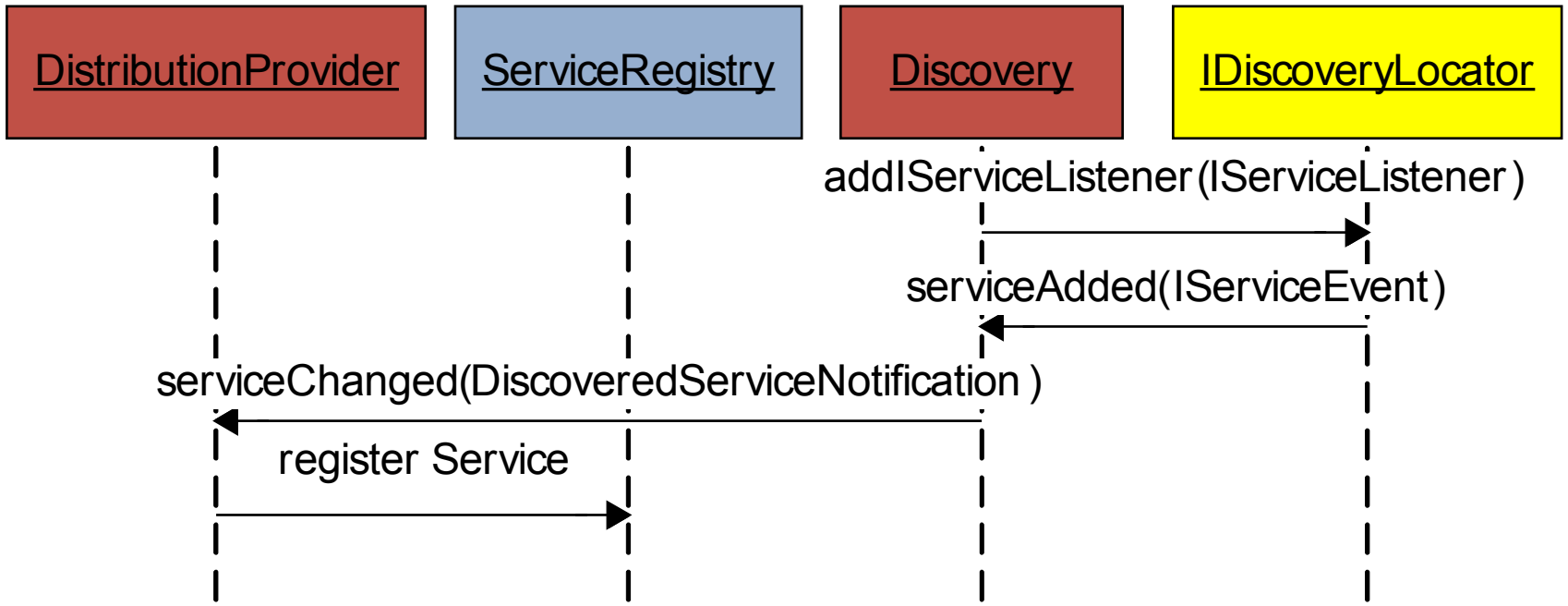
Lab3 – Your own org.osgi.service.discovery.Discovery impl

RFC 119 → ECF Discovery



Lab3 – Your own org.osgi.service.discovery.Discovery impl

ECF Discovery → RFC 119



org.osgi.service.discovery.ServicePublication

org.osgi.service.discovery.ServicePublication is just a „marker“ interface denoting a service for ***org.osgi.service.discovery.Discovery***

- The service will have „***osgi.remote***“ set
- ***Discovery*** is registered as a ***org.osgi.framework.ServiceListener*** or ***ServiceTrackerCustomizer***

Lab3 – Your own org.osgi.service.discovery.Discovery impl

1. Replace ***org.eclipse.ecf.osgi.services.discovery*** with ***org.eclipse.ecf.tutorial.osgi.services.discovery*** in your launch config

1. Implement all methods marked with **//TODO tutorial** in ***org.eclipse.ecf.tutorial.osgi.services.discovery.ServicePublicationHandler***
   - ***Check CVS history if you are stuck***

1. Add ***org.eclipse.ecf.provider.discovery***, ***org.eclipse.ecf.provider.jmdns***, ***org.eclipse.ecf.provider.jslp*** and ***dependencies*** (add required plug-ins) to launch config

1. Fire up ***Discovery and RemoteService UI*** launch

1. Have ***Wireshark*** runnig if you want to know what happens on the wire

1. Run **Lab2**

Service Ranking of IDiscovery* providers

1. CompositeDiscovery (1000)
2. (FilebasedDiscovery ~900)
3. JmDNSDiscovery (750)
4. JSLPDiscovery (500)

But you can also lookup a specific provider explicitly by org.eclipse.ecf.discovery.IDiscoveryLocator.CONTAINER_NAME and org.eclipse.ecf.discovery.IDiscoveryAdvertiser.CONTAINER_NAME

```
aBundeContext.createFilter("(&(" +
   Constants.OBJECTCLASS + "=" +
   IDiscoveryAdvertiser.class.getName() + ")(" +
   IDiscoveryAdvertiser.CONTAINER_NAME +
   "=ecf.discovery.composite))");
```

File based discovery

- **Use Case**: „Client knows the service already“
  - Simplest form of discovery
  - No need for an external discovey mechanism
  - Great for static configuration

File based discovery à la RFC119

```
Bundle-Name: An ECF example
Bundle-SymbolicName:
   o.e.e.tests.osgi.srvc.disc.local.poststarted2
Bundle-Version: 1.0.0.qualifier
Bundle-Vendor: Eclipse.org
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
```

!!! OSGI-INF/remote-service/*.xml is default !!!

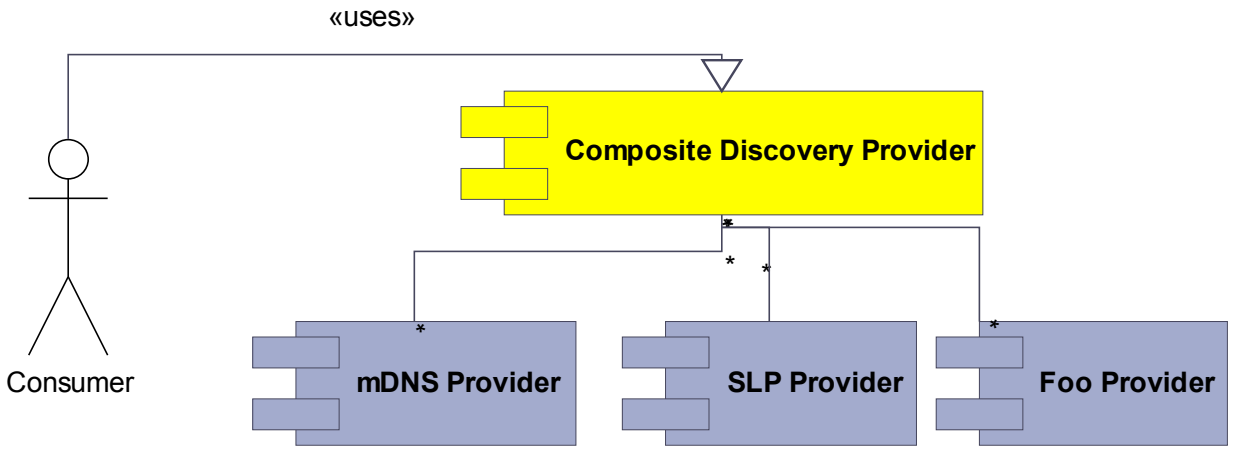
File based discovery à la RFC119

```
<?xml version="1.0" encoding="UTF-8"?>
<service-descriptions xmlns="http://www.osgi.org/xmlns/sd/v1.0.0">
  <service-description>
    <provide
   interface="org.eclipse.ecf.discovery.IDiscoveryAdvertiser"/>
    <property
   name="ecf.sp.cid">org.eclipse.ecf.provider.r_osgi.identity.R_OSGi
   Namespace:r-osgi://localhost:9278</property>
    <property name="ecf.sp.cns">ecf.namespace.r_osgi</property>
  </service-description>
</service-descriptions>

<service-descriptions xmlns="http://www.osgi.org/xmlns/sd/v1.0.0">
  <service-description>
    <provide
   interface="org.eclipse.ecf.discovery.IDiscoveryLocator"/>
</service-description>
</service-descriptions>
```

Composite Discovery Provider

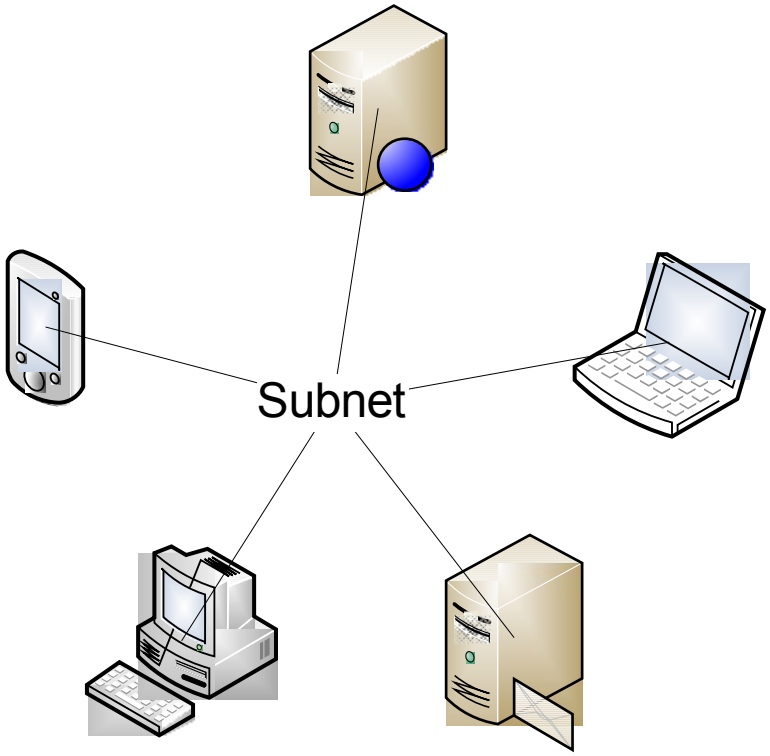
- **Use case**: All (available) discovery providers at once
- While providing the same interface to clients

- Does not filter redundant `IServiceEvents` (yet)
- Dynamic enabled
  - Stores service registrations to reregister with newly added providers



Multicast based providers

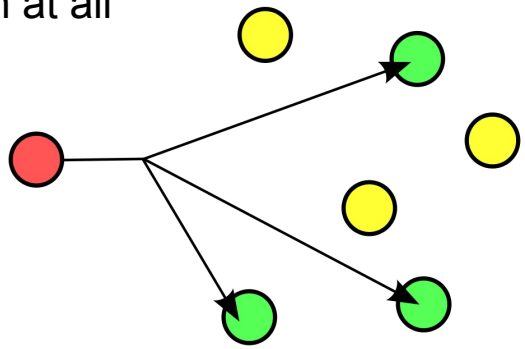
- **Use case**:
  Discovery services in a (highly) dynamic network
  - Even without central (server) infrastructure
  - Peer2Peer



IP Multicast – RFC 1112,1458, 2236…

Recap –

- "*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



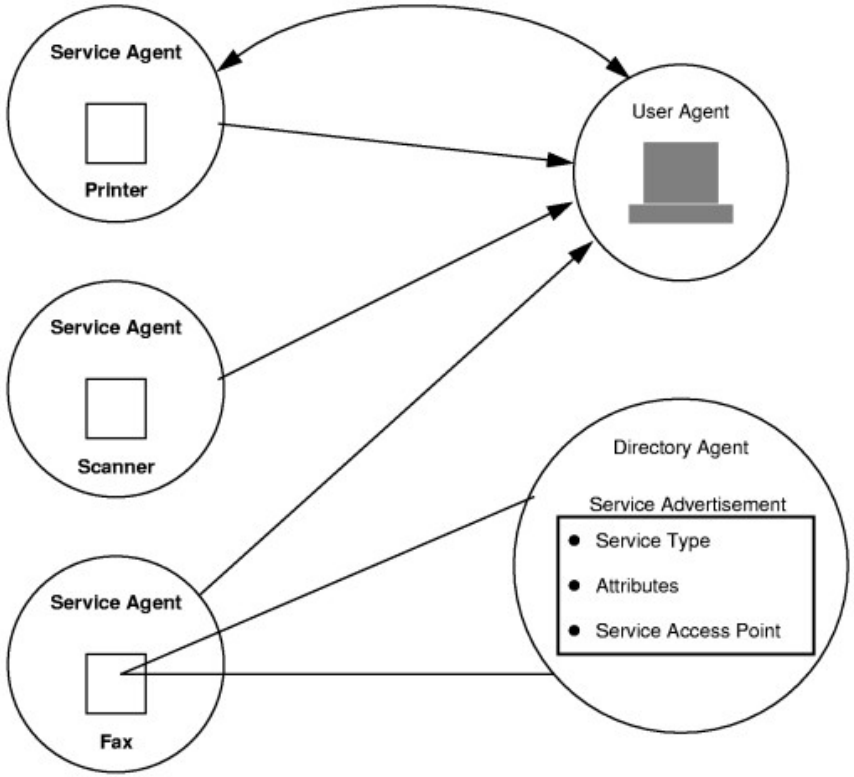
Service Locator Protocol (SLPv2) RFC 2608, …

- Multicast discovery
  - Multicast group 239.255.255.253
    - administratively scoped multicast (RFC 2365)
  - Port 427 (privileged port!)

- User agent (**UA**)
  - maps to Locator
- Service agent (**SA**)
  - maps to Advertiser
- Directory agent (**DA**)
  - Optionally deployed
  - jSLP 2.0
  - OpenSLP

- **Seamless** transition from
  - *Multicast convergence* to
  - *Directory Agent* (DA)
    - DA discovery still multicast

or hard coded



Directory Agents

"DAs cache service location and attribute information. They exist to enhance the performance and scalability of SLP. Multiple DAs provider further scalability and robustness of operation, since they can each store service information for the same SAs , in case one of the DAs fail" [RFC2608]

SAs and UAs fall back to unicast (except for active DA detection)

IP multicast DNS (mDNS)

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

(Incomplete) Comparison of mDNS and SLP

SLP	mDNS
Close(r) to OSGi Services Attributes > Service properties LDAP filters	
Privileged port 427	No privileged port (5353)
Seamless transition from multicast convergence to DAs	Can use existing infrastructure (DNS server) for service discovery, but with a different scope/domain (!= “local”)
	Non-local service registration via *DNS Dynamic Update*
Security based on public keys	Security based on *DNSSEC*
Reactive	Proactive
Java.lang.String > SLP strings	
Run your own naming authority	Centrally managed list of service types (See http://www.dns-sd.org/ServiceTypes.html)
No method to lookup all scopes (See https://bugs.eclipse.org/218308)	

Other service discovery protocols

- Based on Multicast
  - Simple ServiceDiscovery Protocol (SSDP) used in UPnP
  - Jini
- Universal Description, Discovery and Integration (UDDI)
- Bluetooth Service Discovery Protocol (SDP)
- Salutation (disbanded ~2005)
- XMPP Service Discovery (XEP-0030)
- Web Services Dynamic Discovery
- …

How to handle port conflicts?

In case the “official” port is blocked by another application (e.g. Apple’s mdnsResponder) or fails to bind for another reason (e.g. privileged port <1024)

For JmDNS:

“*-Dnet.mdns.port=65353*”

For jSLP:

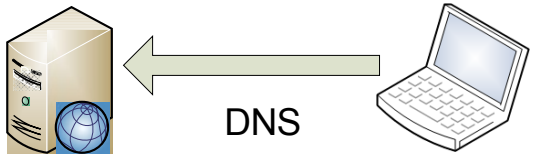
“*-Dnet.slp.port=65427*”

Meet a hackish multicast port redirector (and make sure to run it only on one host pro network)

```
public void run() {
    while(isRunning) {
        try {
         byte buf[] = new byte[maxPacketSize];
         DatagramPacket packet =
              new DatagramPacket(buf, buf.length);
         from.receive(packet);

         int srcPort = packet.getPort();
         InetAddress srcAddr = packet.getAddress();

         // don't create a loop
         if (partnerThread.getLocalPort() == srcPort
              && localhost.equals(srcAddr)) {
              continue;
         } else {
              packet.setAddress(multicastGroup);
    packet.setPort(dstPort);
              to.send(packet);
      } catch (IOException e) {
         e.printStackTrace();
    }
}
```

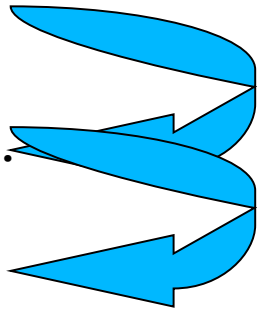
DNS-SD

- **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 with Bind

```
@ 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.

_http._tcp IN TXT "path=/ecf"
_http._tcp IN TXT "dns-sd.ptcl=http"

…
www.eclipse.org. IN A 123.123.123.123
```



Lab4 – Your own org.eclipse.ecf.discovery.IDiscoveryLocator

1. Implement all methods marked with **`//TODO tutorial`** in ***`org.eclipse.ecf.tutorial.provider.discovery.dnssd.DnsSdDiscoveryLocator`***
   - ***Check CVS history if you are stuck***
   - ***TDD with Junit tests in org.eclipse.ecf.tests.provider.dnssd***

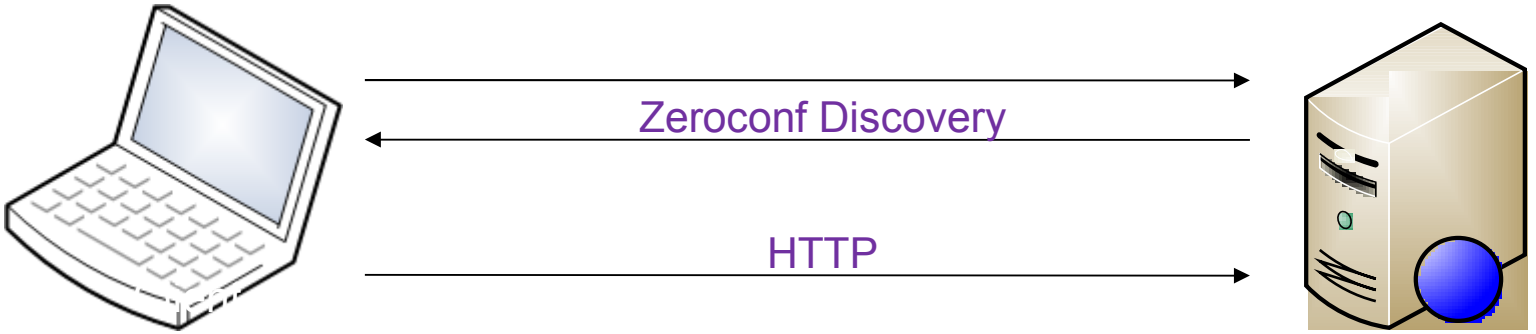
1. Add ***`org.eclipse.ecf.tutorial.provider.discovery`*** to launch config

1. Fire up ***`Discovery and RemoteService UI`*** launch

1. Run **Lab3**

Instead of 1. and 2. feel free to implement your very own discovery locator/ advertiser based on e.g. UDDI, JINI, whatnot. We will try to help. ☺

Demo

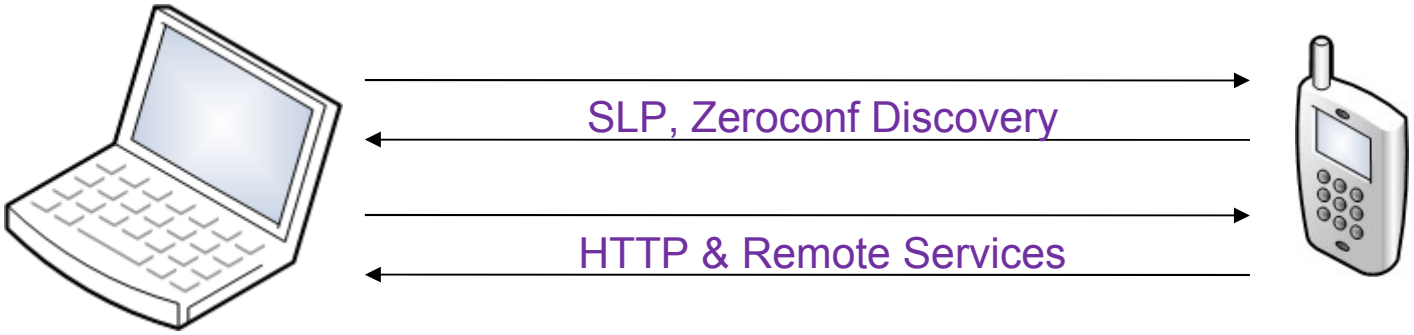


running RCP App with ECF Discovery (using SLP and mDNS providers) and ECF Remote Services (using R-OSGi Provider)

Apache HTTP Server
Either avahi or mod_dnssd

supports Zeroconf

Demo



iPhone

running RCP App with ECF Discovery (using SLP and mDNS providers) and ECF Remote Services (using R-OSGi Provider)

running Equinox and ECF, featuring Knopflerfish HTTPConsole and HTTP based directory listing servlet

ECF Distributed Testing

Distributed Testing with ECF

- ECF does not offer a solution for Distributed Testing (yet)
  - Will get focus past Galileo
- Open discussion on distributed testing
- Collaborative exploration of ideas, design, use cases, requirements

Lessons learned

References

- http://files.dns-sd.org/draft-cheshire-dnsext-dns-sd.txt
- http://files.multicastdns.org/draft-cheshire-dnsext-multicastdns.txt

More about distributed OSGi at EclipseCon

- "*Real world distributed OSGi with Paremus Service Fabric*"
  http://www.eclipsecon.org/2009/sessions?id=828

- "*Distributed OSGi Demo*"
http://www.eclipsecon.org/2009/sessions?id=251

- "*Best Practices for Distributed OSGi Services*"
  http://www.eclipsecon.org/2009/sessions?id=633

- "*Distributed OSGi Services*"
http://www.eclipsecon.org/2009/sessions?id=757

- "*Distributed OSGi*"
- http://www.eclipsecon.org/2009/sessions?id=756

- … just the ones that have "distributed" on their title

More information on distributed OSGi

- http://www.osgi.org/download/osgi-4.2-early-draft2.pdf
- http://www.osgi.org/Specifications
- Reference Implementation based on Apache CFX
  - http://cwiki.apache.org/confluence/display/CXF/Distributed+OSGi

Eclipse ECF Project

http://www.eclipse.org/ecf
http://wiki.eclipse.org/ECF

ECF 3.0 will ship with Eclipse Galileo

Questions?

Legal Notices

- 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.