Best Practices for Distributed OSGi Services

Markus Alexander Kuppe
Scott Lewis

http://www.eclipse.org/ecf

"*Isn't D-OSGi... just repeating the mistake of every other RPC-based system in the last 20 years? Discuss...*"

http://twitter.com/njbartlett

"Isn't D-OSGi... just repeating the mistake of every other RPC-based system in the last 20 years?

NO!*

*It's our job to make sure of this...and the job's not finished

D-OSGi requirements & assumptions

- Keep the current (OSGi) programming model where possible
- Abstraction from protocol, communication, data implementation
- Allow interop with non-OSGi systems in heterogeneous environments
- Allow clients running outside of OSGi to discover services
  - But use of org.osgi.service.discovery.Discovery is optional
- Bring service oriented programming model to distributed computing
- Much of the problems of distributed systems are already covered by the dynamic nature of OSGi services
  - ServiceException has new type “REMOTE”

Network Transparency

- Do you hide the network's aspects from the programmer?
  - Performance/Timing
  - Reliability/Partial Failure
- **Not** a good idea
  - A Note on Distributed Computing
  - Lots of failed attempts at doing so - See Neil's tweet

One conclusion: Eventually...programmers of distributed service clients **want to know** and **react** to what's happening

Enter: OSGi Services

Now have OSGi Service Registry/ServiceReference/ServiceRegistration

- The framework manages service registration/lookup/and cleanup
  - bundleContext.registerService(...)
  - bundleContext.getServiceReferences(...)
  - bundleContext.stop()

Makes Registration, Lookup, Clean-up **work**

RFC 119 provide **transparent** remote service registration, lookup, and clean-up

This is GOOD Transparency

- Makes it very easy for programmers to use - GOOD
- Providers (implementations) take care of the hard stuff (distribution, serialization, etc)
- Framework handles dynamic services **already** so network services are accomodated already
  - Caveat Emptor: Service programmers **must** make their bundles/services dynamic-aware

But there is also **Usage** Transparency

- Public MyResult foo(MyParameter);
- Clients will expect it to work when they call it
- It's going to fail with RuntimeException...or worse, **block**
- This **will** be much more frequent than local services
- What are clients to do?

Usage Transparency...**Still a problem for those that design service interface**

Transparent **Usage** (cont)

ECF

- Exposes IRemoteService via service property osgi.remote
- Gives proxy AND additional calling patterns to service consumer
- AsyncExec, Future, One-Way

- RemoteServiceTracker
  - Same Functionality as ServiceTracker
  - IRemoteService rather than Object

So What are We Saying?

Separate (at bundle level) interface and implementation

Define your service interface 'carefully'

- Can your clients depend upon your contract?
- Complex objects...serialized? Pass By Reference?

Same local/remote service interface?

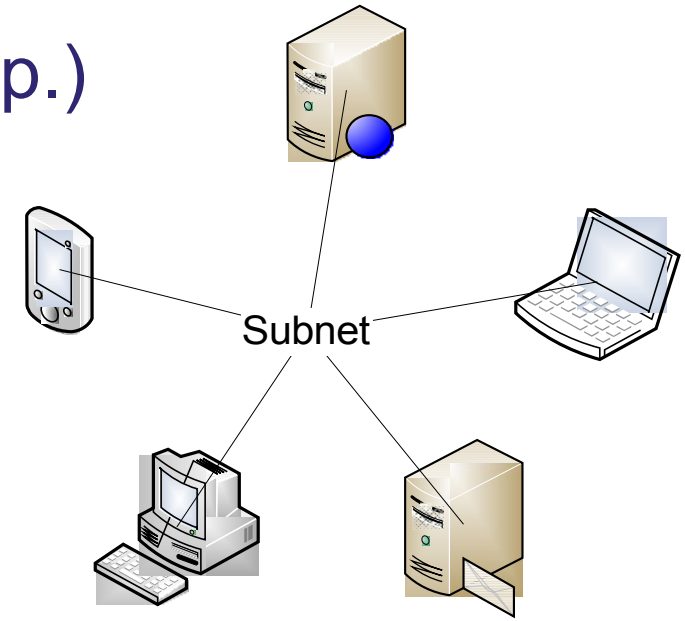
Think about clients (at runtime)

How can/will they respond to failure, blocking, etc

Consider Being More Asynchronous

ECF IRemoteService ...or write your own

Discovery best practice (b.p.)



Discovery best practice for service consumers

NONE!*

Cannot run discovery synchronously upon *getServiceReferences*() as it violates the non-blocking nature of OSGi

*It's our job to make sure of this...and it's done :-)

Discovery best practice for service providers

NONE!*

However, what's with non 119 relevant service props?
...security, marshalling

*It's our job to make sure of this...and it's nearly done :-)

For service providers or consumer in non-OSGi based systems and deployers

- You ignore discovery entirely (just static configuration) :-)
- Find a way to Integrate with existing solutions
  - SLP, mDNS, DNS-SD, UDDI, JINI, … proprietary
- Chose a protocol that best fits your requirements
- Deal with all the protocol, network... details
  - Have network specialist/administrators on the team
- Do not trust service discovery events (unless you secured it)

RFC 119v2 outlook

- Different DSW in one runtime all handle a service registered event?
  - What is the discovery provider supposed to do? Only handle service publication for its matching DSW? For all?
- How to do authentication?
  - Make services available that won't be consumable because of access restrictions
- Being more asynchronous in distributed OSGi as well as the framework itself?

Thanks spec writers
Thanks Scott Rosenbaum for making this talk happen

Questions?

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

Legal Notices

- EclipseSource logo and trademarks are registered
- 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.