Distributed OSGi in Runtime Project

Scott Lewis
EclipseSource, Inc.
Eclipse Communication Framework

http://www.eclipse.org/ecf

Use Cases for Distributed OSGi Services

- Remote Provisioning (Jeff McAffer)
  - http://eclipsesource.com/blogs/2009/05/05/remote-provisioning-wit
  - Remote access/use of p2 engine, planner, profileRegistry
  - Server-managed profiles
- Load Balancing OSGi Services
  - Load balancing **service invocations/method calls** among target servers
  - EclipseSource Yoxos Enterprise: OSGi server that creates Eclipse configurations
    - Distributes P2 planner requests among set of planner servers
  - Can reuse same infrastructure for **any** OSGi service
- Remote Services + Declarative Services
  - Dependency injection and remote services...nice way to build SOA servers

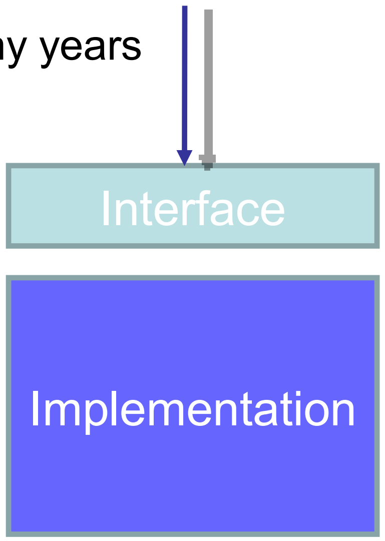
Lots of others

- Modular webservers

OSGi Services: Just Works

- OSGi services provide
  - Encapsulation
  - Loose coupling
  - Extensibility and Abstraction

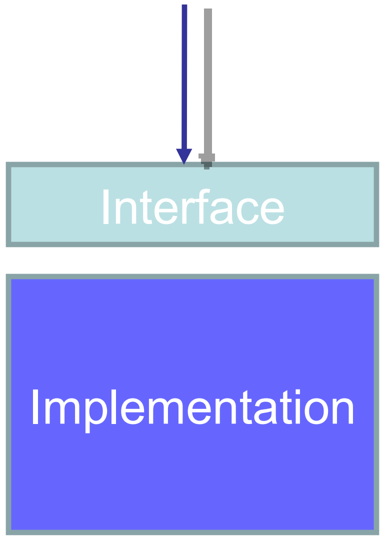
- **Well-worn**: In place in OSGi spec and functioning impls for many years
  - Extensibility Underneath Eclipse
    - Implements Eclipse plugin model
    - Implements Eclipse extension registry
  - Present in commercial app servers
    - BEA Weblogic (now Oracle)
    - IBM Websphere
    - Others
- Great Intro/Description of OSGi services
  - Symmetric Service Oriented Programming (BJ Hargrave) : http://live.eclipse.org/node/778



How are OSGi Services Exposed and Used?

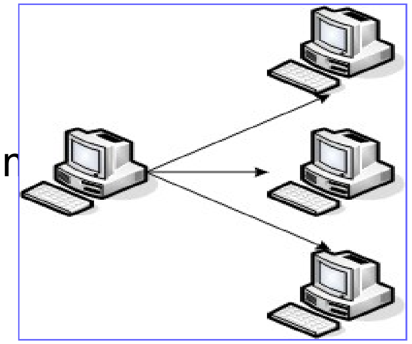
- Registration - 'service host'
  - BundleContext.registerService(...)
- Lookup/Finding - 'service consumer'
  - BundleContext.getServiceReferences(...)
  - BundleContext.getService(ref)
- Use (consumer)
  - Call methods on interface
    - Implementing object's code is synchronously run
- Clean-up
 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

Code example



Distributed OSGi == OSGi Services Over Network

- Registration
  - **New step**: publication for network discovery
- Lookup
  - Network discovery for finding service (not all services are 'just there')
- Cleanup
  - Guarantee cleanup in unreliable network
- Other things unique to remote services
  - Marshalling/Serialization issues
  - call by reference vs. call by value
  - failure handling
  - Rpc (blocking/synchronous calls) in failure-prone environn



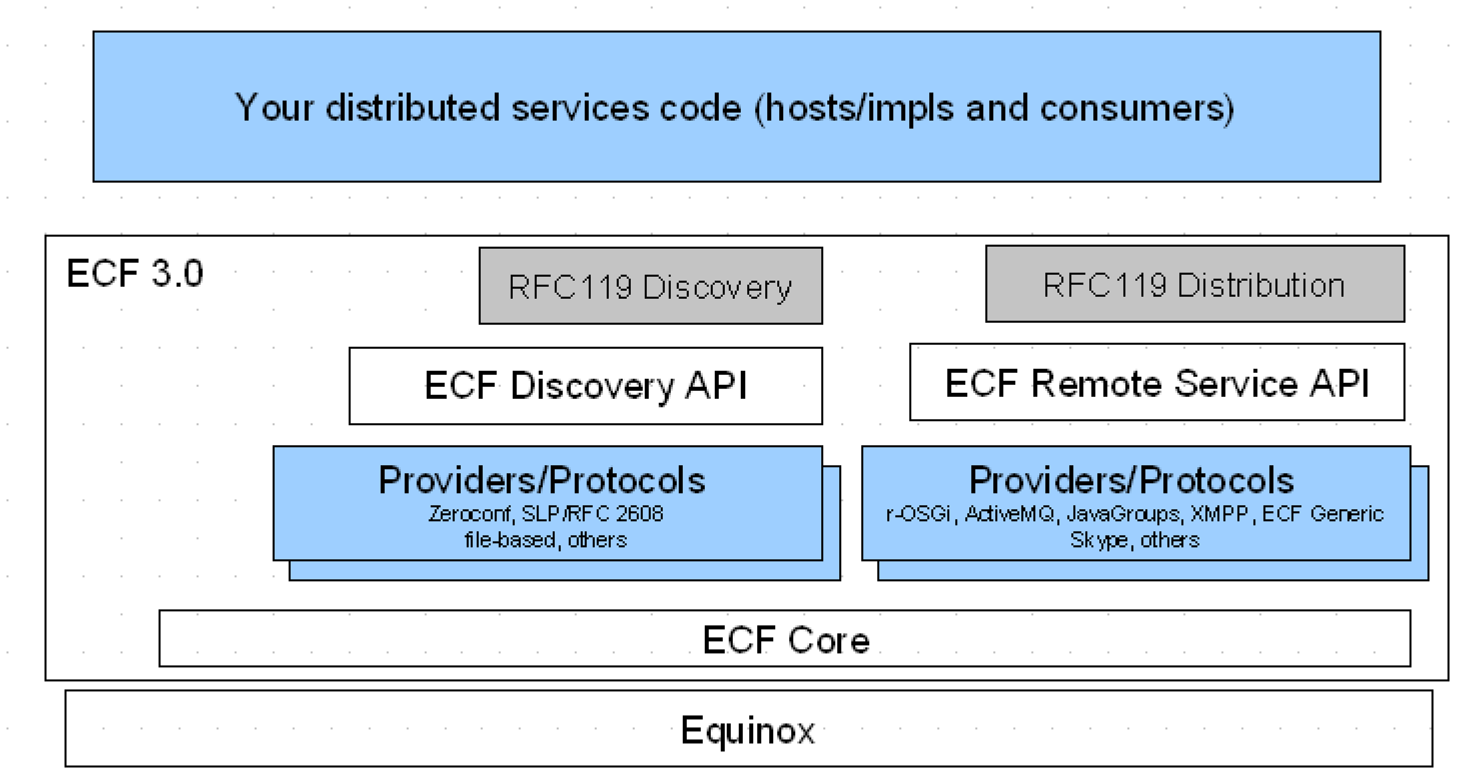
Distributed OSGi Specification: RFC119

- OSGi Alliance Enterprise Experts (EE) Group
- OSGi 4.2 (Early 2010 finalization)
- Drafts available now: http://www.osgi.org/Specifications/Drafts
- ECF 3.0/Galileo Implemented Draft Equinox Runtime
- Will have continue to have spec-compliant implementations
  - Scott not spec author (focus on impls in Equinox)
  - Frequently complains (aka contributes) to spec authors
  - Jan Rellermeyer...r-OSGi creator and ECF committer...is in EE working group
  - ECF's remote services API has features not yet in RFC119
    - Asynchronous messaging support (e.g. futures and listeners)

Distributed OSGi - Architecture

- Subsystems
  - Discovery
    - Service Publication (service host) and discovery (consumer)
  - Distribution
    - Creation/access to proxy
    - Marshalling/Serialization (e.g. parameters)
    - Currently: Spec defines proxied RPC...i.e. blocking call/return
- Spec does **not** define wire protocol for either discovery or distribution
  - Allows multiple protocols to be accomodated

ECF 3.0/Galileo Implementation



ECF Remote Services

- Protocol agnostic API for doing remoting
- Like Local OSGi Services
  - Registration
    - registerRemoteService(...)
  - Lookup
    - IRemoteServiceReference ref = getRemoteServiceReferences(...)
  - Cleanup
    - unregister/unget
- **Used** ECF Discovery and Remote Services API to implement RFC 119
  - Programmer can use transparent registration/lookup from RFC119
  - **Or** non-transparent registration/lookup via ECF RS

ECF Remote Services Providers

- R-OSGi
- ECF Generic
- XMPP
- JMS (ActiveMQ but other JMS impls trivial to create)
- Skype
- JavaGroups
- REST: Google SOC 2009 project
  http://wiki.eclipse.org/Google_Summer_of_Code_ECF_Projects_for_2009
- Others pending: Riena, SOAP-based, commercial/closed, CXF, etc
- Any/all ECF RS providers get ongoing, **free** support for RFC 119

ECF Discovery API

- A protocol agnostic API for service discovery
  - Does not expose provider/protocol internals
    - Allows flexibility in service addressing
  - Not limited to the local subnet (LAN)
    - However some providers/protocols 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

ECF Discovery Providers

- Zeroconf/Bonjour
- Service Locator Protocol (IETF RFC2608)
- Composite provider
- File-based discovery (xml)
- Others being created
  - private/commercial
  - Other protocols...e.g. XMPP discovery, UPnP

RFC 119

- Transparent Registration, Lookup, and Cleanup
  - BundleContext.registerService and BundleContext.getServiceReferences
  - Don't have to be concerned with details of publication/discovery, network lookup, proxy creation, etc
- 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

Do we want network transparency?

- 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
    - OSGi's Service Model is Dynamic
    - Consumers must deal with services coming and going in regular OSGi services
    - Can use dynamic services to represent network failure (service disappears)
    - We want to allow/support programmers in using both transparent and non-transparent usage

ECF → Transparent **or** non-transparent remoting

- Exposes IRemoteService
- Provides proxy AND other calling patterns to consumer

AsyncExec, Future, One-Way all available

  - AsyncExec, Future, One-Way all available
  - RemoteServiceTracker (ServiceTracker for IRemoteServices)
  - IRemoteService getRemoteService() rather than Object getService()

Other exciting things in ECF project

- GSOC: REST provider support
  - http://wiki.eclipse.org/REST_abstraction_for_ECF
- GSOC 2009: SIP/VOIP
  - http://socghop.appspot.com/student_project/show/google/gsoc2009/ec
- GSOC 2009: Google Services Provider
  - http://socghop.appspot.com/student_project/show/google/gsoc2009/ec
- GSOC 2009: Distributed testing framework
  - Testing Distributed OSGi – based apps
- Real-time Shared Editing
  - EclipseDay talk last year: http://live.eclipse.org/node/543
- ECF Provider for Google Wave
  - http://eclipsesource.com/blogs/2009/06/15/google-wave-and-e
  - Implement Google Wave server-to-server protocol as ECF provider

Reference Distributed OSGi links from EclipseCon 2009

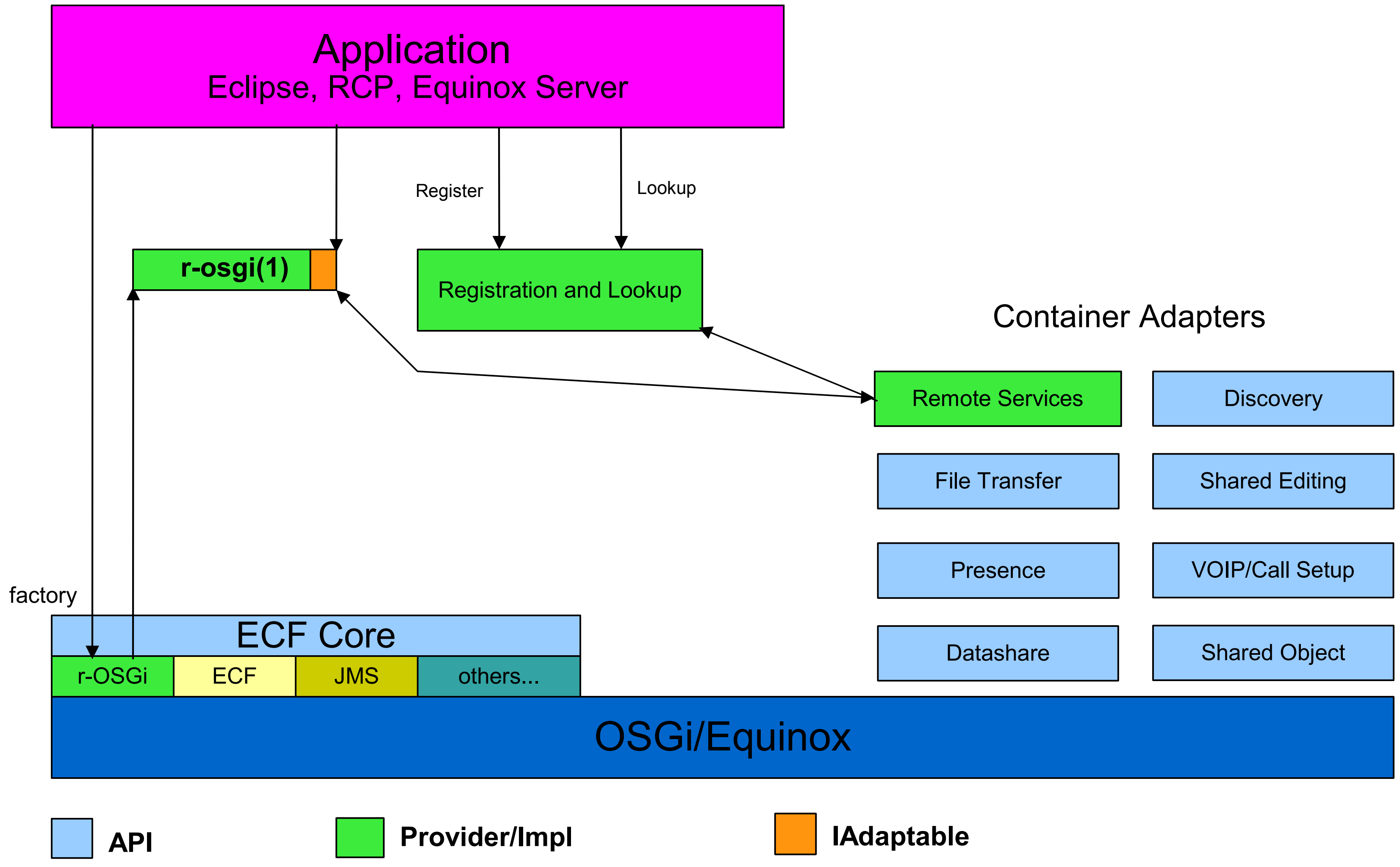
- “*Distributed OSGi Demo*”
http://www.eclipsecon.org/2009/sessions?id=251

- “*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

ECF Provider Architecture



File based discovery

```
<?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>
```

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
```