Introduction to Equinox and OSGi:
Building Blocks for Applications

Thomas Watson, IBM Lotus
Equinox Project Co-lead

Why is Equinox interesting?

- Component Orientated Development and Assembly
  - Develop self-contained components
  - Make it easy to integrate and extend components
  - Allow for the assembly of components to create solutions
- Cross Platform and Domain
  - Supports a wide variety of hardware and operating systems
  - Applicable to many domains – Embedded, Desktop, Server
  - Reusable components
    - Reuse of skills and technology
    - Innovation through integration
- Standards compliant runtime – OSGi

What gives Equinox its power?

- OSGi Technology
  - Based on OSGi Alliance Service Platform Specification R4.2
- Component Oriented
  - Building runtimes requires componentization
  - Modules are packaged as self-describing bundles
  - Strong notion of versions is built into the Framework
- Dynamic / Lifecycle
  - Bundles can be installed, started, stopped, uninstalled at any time
- Collaboration Facilities
  - Service Orientated. Provides a familiar publish/find/subscribe model for service objects within a given runtime
  - Extensibility. Allows for component customization and extension

Benefits of Equinox

- Standardized component model for code packaging and collaboration
- Micro-kernel approach – start small, grow as needed
- Open Standard and Open Source avoids proprietary and custom lock-in
- Flexibility to change and re-use of components
  - Improve developer productivity
  - Consistent programming model across environments
  - Achieve faster entry into new markets
- Allow for the assembly of components to create solutions for an agile IT

OSGi Technology

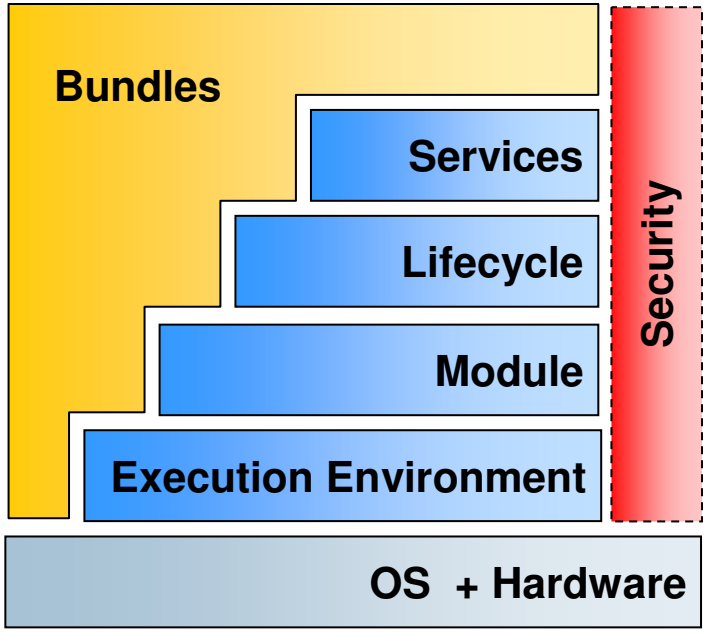
The Dynamic Module System For Java™ Platforms

OSGi Technology

- Specifications developed by the OSGi Alliance
  - Member companies collaborate on developing open standards
  - The technology applies to a wide range of domains
    - Embedded, Desktop, Server …
  - Platform agnostic
- Equinox team has a high level of involvement in OSGi
  - Developing the specification
  - Developing some of the reference implementations
  - Developing some of the Compliance Tests
- Expert groups
  - Core Platform Expert Group – Core Framework and services
  - Mobile Expert Group
  - Enterprise Expert Group

OSGi Technology

- The Framework is split up into different layers
  - *Execution Environmemt* – the VM
  - *Module Layer* – Module system for the Java Platform
  - *Lifecycle Layer* – Dynamic support
  - *Service Layer* – Module collaboration



Wide Spread Adoption of OSGi and Equinox

- Many framework implementations
  - Equinox – Open source
  - Felix – Open source
  - Knopflerfish – Open source
  - Concierge – Open source
  - ProSyst
  - …
- Interest from the Enterprise Space
  - Many EclipseRT projects (Jetty, Swordfish, ECF, EclipseLink, RAP)
  - Spring Dynamic Modules for OSGi
  - Apache Aries Project
- All Eclipse-based systems run on Equinox
  - Runtimes (e.g., RAP, Swordfish, Riena, ECF, EclipseLink)
  - RCP, eRCP
  - Tooling

Equinox is Used Across a Wide Range

- Equinox OSGi as a component runtime
- Consistent programming model from embedded to server
- Reuse components across the spectrum
- Some examples…

Equinox Building Blocks

Equinox Building Blocks

OSGi Standards

- Equinox OSGi Framework
  - Module Runtime
  - Bundle Lifecycle
  - OSGi Service registry
- Application Admin Service
- Configuration Admin Service
- Device Access
- Declarative Services
- Event Admin Service
- Http Service
  - Tiny and Jetty
- IO Connector Service
- Log Service
- Metatype Service
- Preferences Service
- User Admin Service
- Wire Admin Service

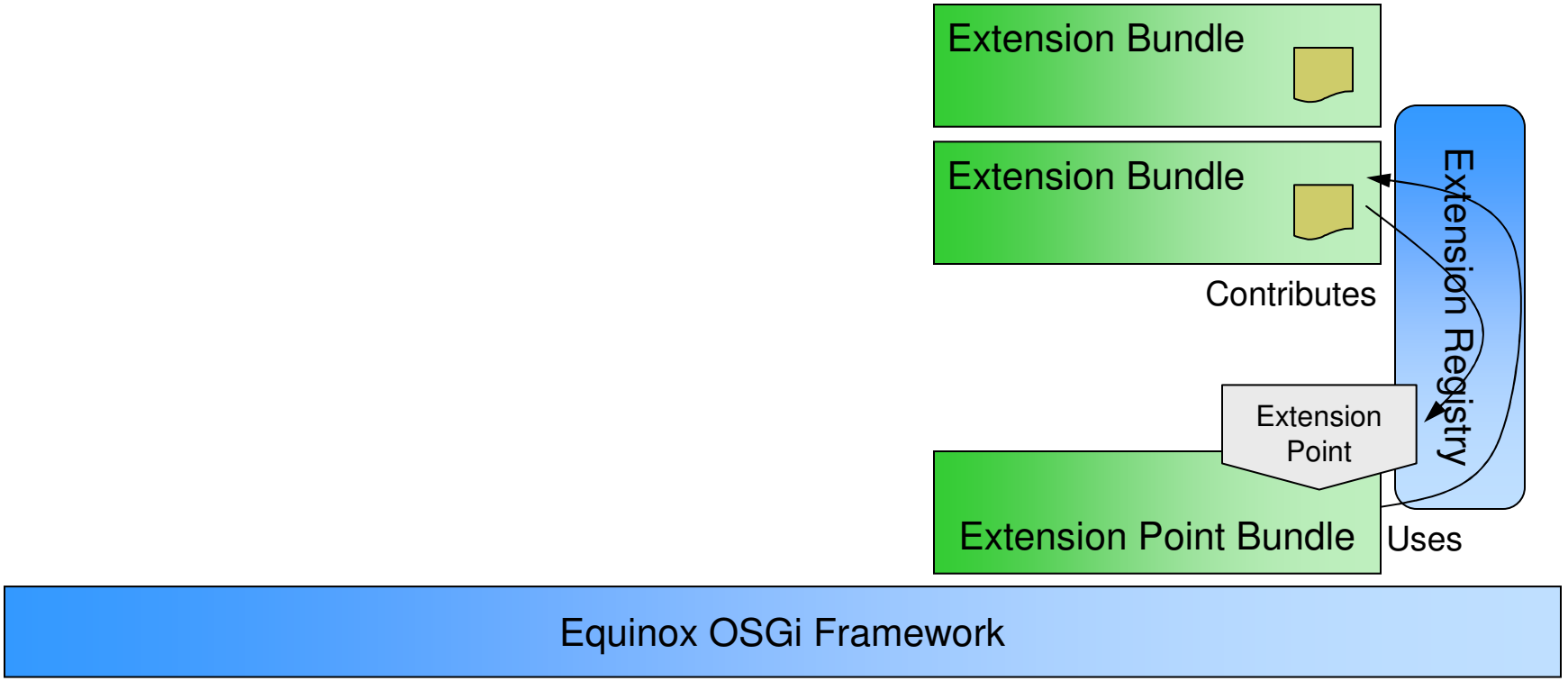
Equinox Enhancement

- Native Launcher
- Splash Support
- Eclipse Extension Registry
- Eclipse Application Container
- Equinox Server-Side
- Provisioning with p2
- Equinox Security
- Buddy Class Loading
- Framework Adaptor Hooks
- Service Activator Toolkit

Bundle Collaboration

- Two complementary mechanisms
  - Services
  - Extensions
- Common attributes for collaboration in Equinox
  - Dynamic – participants can come and go
  - Tracking facilities
  - Declarative and programmatic collaboration
- Differences the two
  - Contract – How to defined, who implements and who consumes
  - Lifecycle – When contributions are available, can be used

Extensions

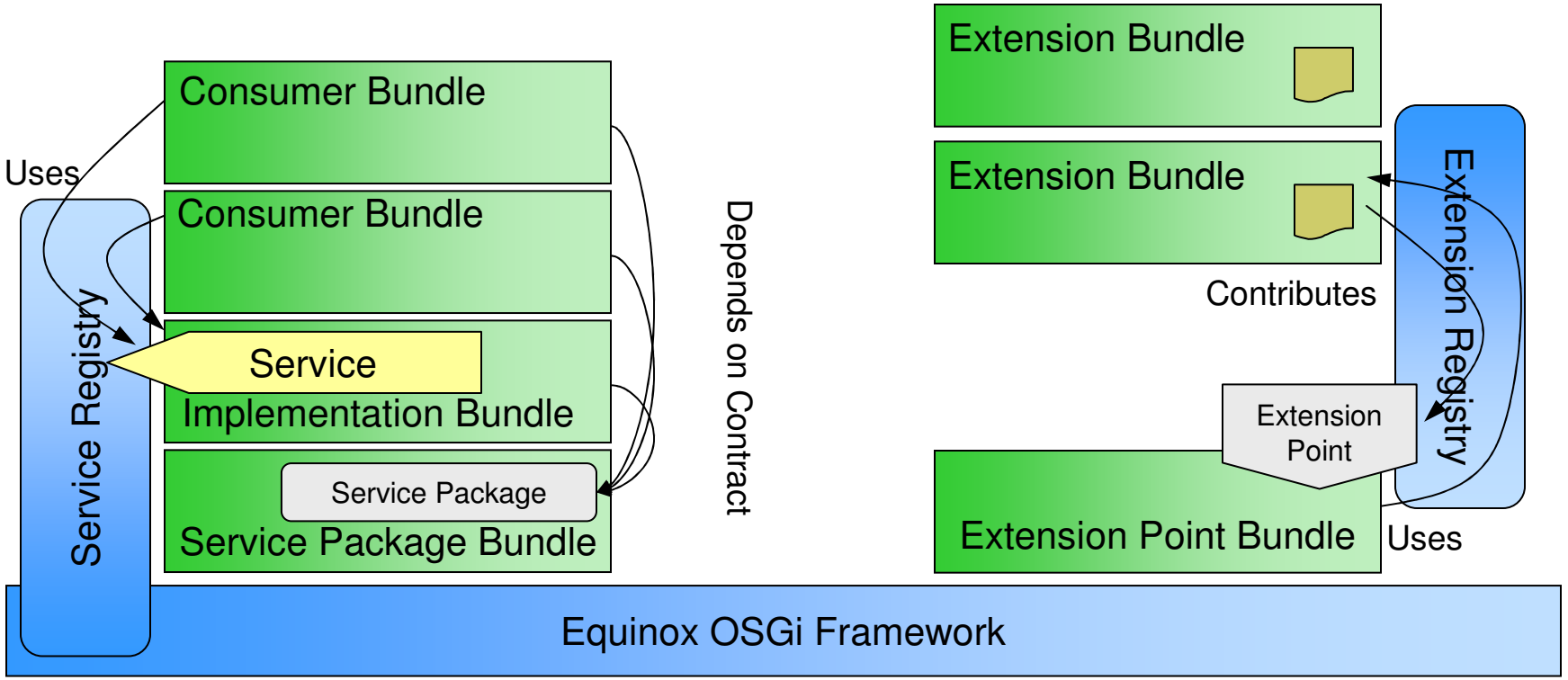


Bundle Collaboration – Extension Registry

- Bundle defines contract via Extension Points
  - Contract is declared in *plugin.xml*
  - May involve Java API and/or additional data (e.g. static help content)
- Contract fulfilled by contributing Extensions
  - Extensions are declared in *plugin.xml*
  - Supplies required data and concrete implementations of Java API
- Contract consumer (typically contract definer)
  - Extension point provider consumes extensions
  - Extensions are lazily loaded as needed by consumer
- Lifecycle
  - Bundle RESOLVED event or code triggers collaboration
  - Resolution state cached – quick re-launch with 1000s configured

Services

Extensions

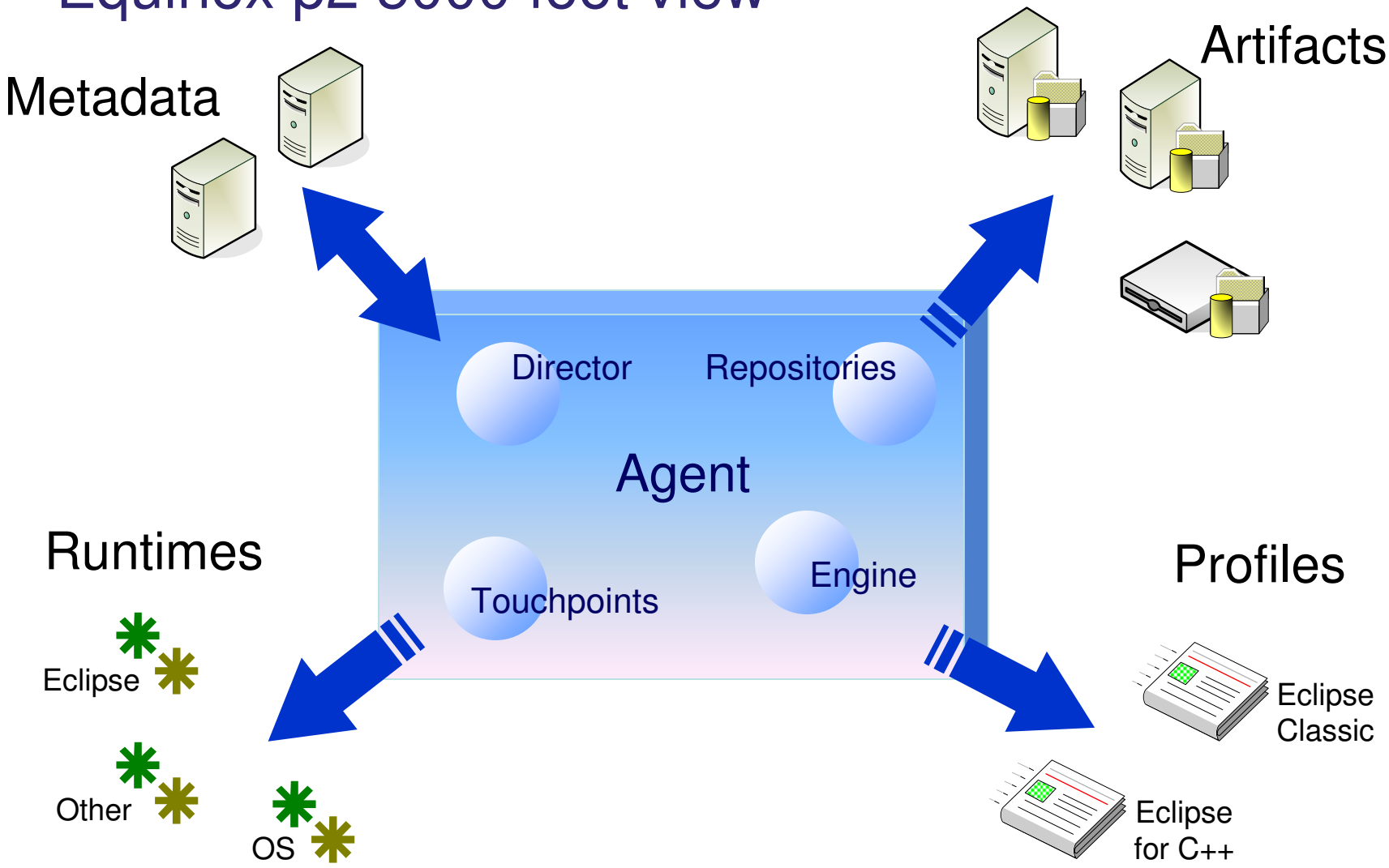


 |

Bundle Collaboration – OSGi Services

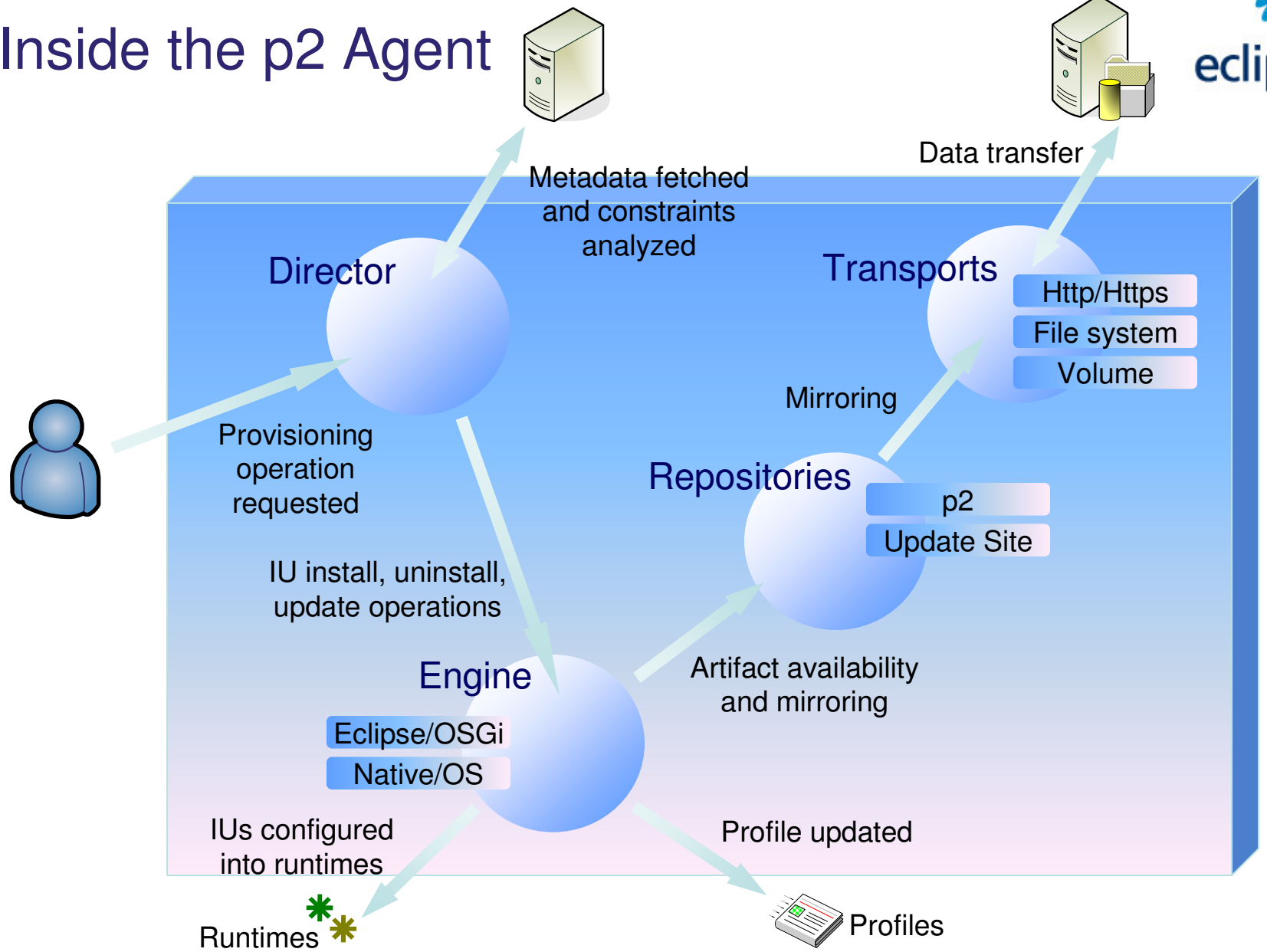
- Contract defined by Java interface or class
  - Service contract shared across multiple consumers and producers
  - Contract is based on Java API
- Contract is fulfilled by service Implementers
  - Provide a concrete implementation of service contract
  - Declarative or programmatic service implementation registration
- Contract consumers
  - Discover/Track available services and get instances
  - Service objects consumed by any bundle in the system
- Lifecycle
  - Producers and consumers of services must be running
  - Event model and lifecycle inhibits caching

Equinox p2 5000 foot view



 |

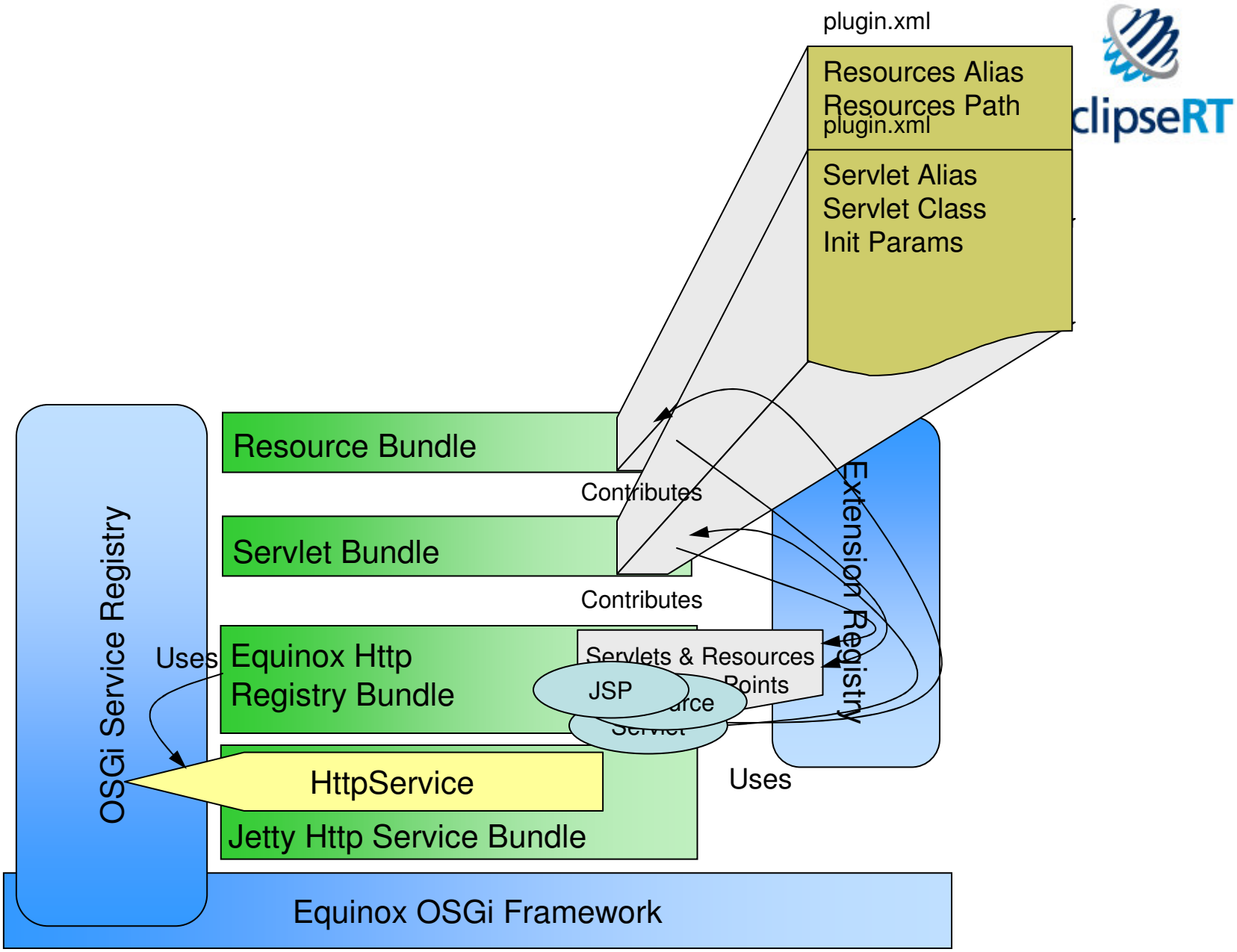
Inside the p2 Agent



 |

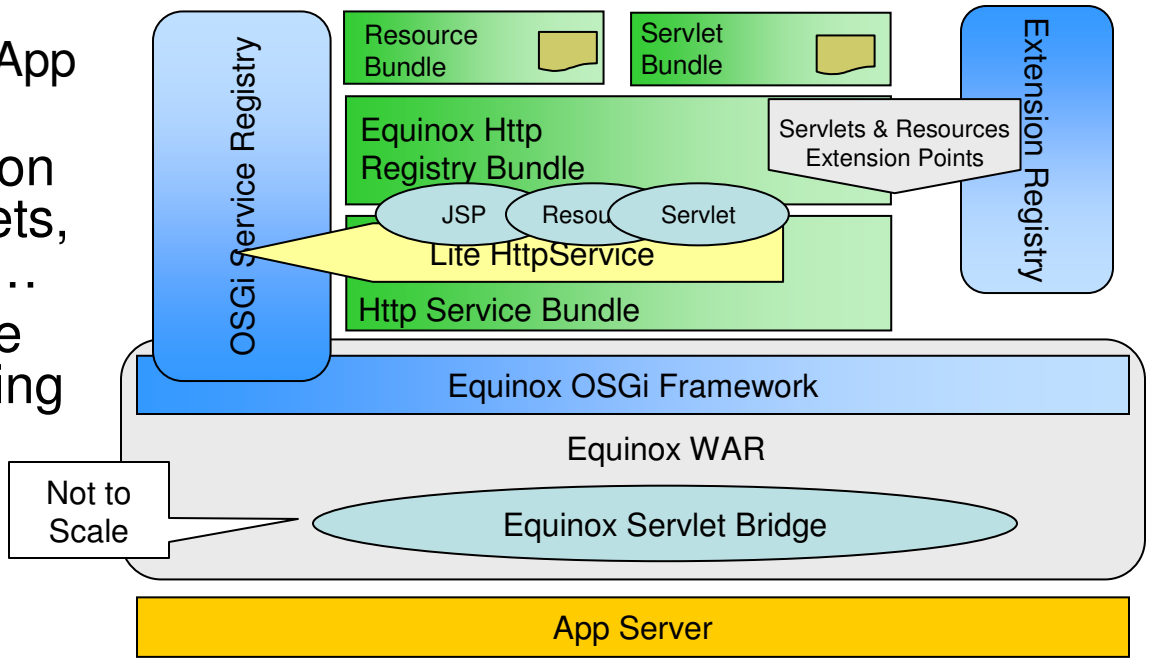
Equinox Server-Side

- Based on OSGi HTTP Service Specification
  - Code-based registration of servlets and resources
  - Two implementations available
    - Lightweight – geared towards embedded devices
    - Jetty-based – full-featured, more robust, performant, …
- Equinox Additions
  - Servlet Bridge
  - JSP 2.0 support
  - Improvements to the HTTP Service's Servlet API support
    - File extension support for URI mappings (e.g. /*.jsp)
  - Contribution of content via the Extension Registry
- Another example of extensions and services working together!!



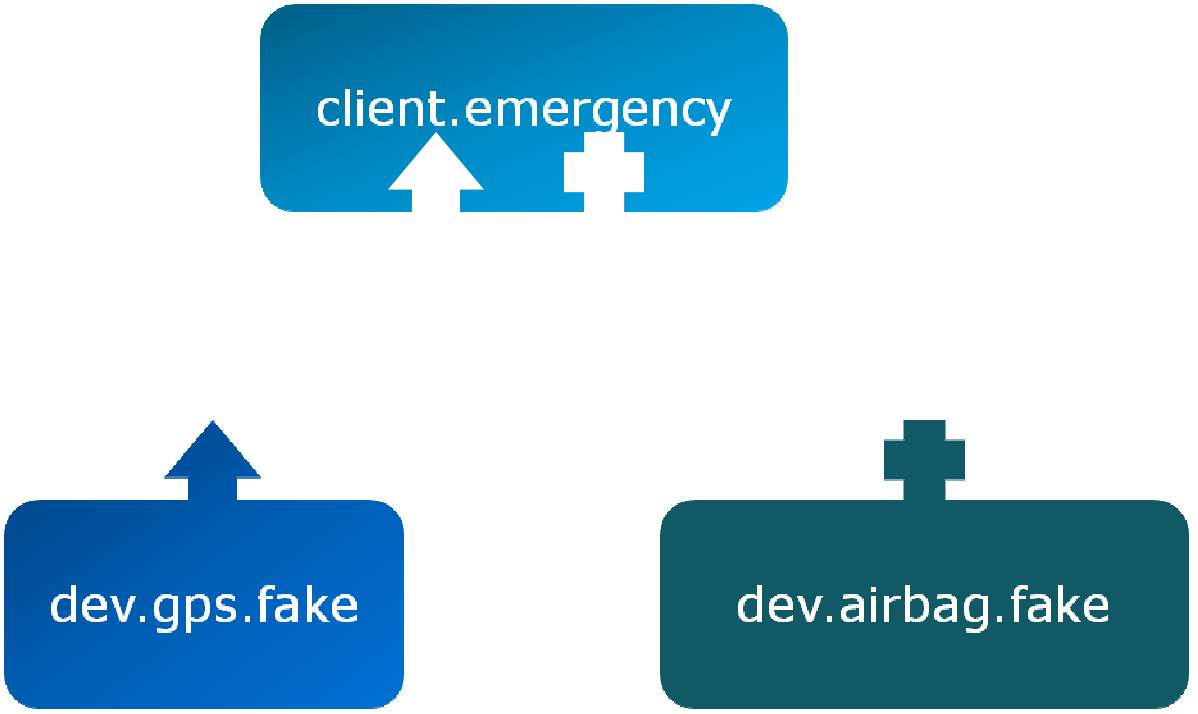
Equinox Server-Side in an App Server

- Bridge servlet hosts Equinox in traditional App Server
- Application isolation
- Integration with existing infrastructure
- Lite HTTP Service
  - Expose underlying App Server capabilities
- Add application function as bundles with servlets, JSPs, static content, …
- Install/Update/Manage application by managing bundles



Equinox Example

Embrace Dynamism



Dangers of Service Tracking

Activator

```
public class Activator implements BundleActivator {
  private BundleContext context;
  private EmergencyMonitor monitor;
  private ServiceTracker gpsTracker;
  private IGps gps;
  private ServiceTracker airbagTracker;
  private IAirbag airbag;

  public void start(BundleContext context) throws Exception {
    this.context = context;
    monitor = new EmergencyMonitor();

    // Start tracking IGps services.
    ServiceTrackerCustomizer gpsCustomizer =
        createGpsCustomizer();
    gpsTracker = new ServiceTracker(context,
        IGps.class.getName(),
      gpsCustomizer);
    gpsTracker.open();

    // Start tracking IAirbag services.
    ServiceTrackerCustomizer airbagCustomizer =
      createAirbagCustomizer();
    airbagTracker = new ServiceTracker(context,
        IAirbag.class.getName(),
      airbagCustomizer);
    airbagTracker.open();
  }

  public void stop(BundleContext context) throws Exception {
    // Stop tracking IAirbag services.
    airbagTracker.close();

    // Stop tracking IGps services.
    gpsTracker.close();
  }

  private ServiceTrackerCustomizer createGpsCustomizer() {
    return new ServiceTrackerCustomizer() {
      public Object addingService(ServiceReference reference) {
        Object service = context.getService(reference);
        synchronized (this) {
          if (Activator.this.gps == null) {
            Activator.this.gps = (IGps) service;
            Activator.this.bind();
          }
        }
        return service;
      }

      public void removedService(
          ServiceReference reference, Object service) {
        synchronized (this) {
          if (service != Activator.this.gps)
            return;
          Activator.this.unbind();
          Activator.this.bind();
        }
      }

      public void modifiedService(ServiceReference reference,
          Object service) {
        // No service property modifications to handle.
      }
    };
  }

  private ServiceTrackerCustomizer createAirbagCustomizer() {
    return new ServiceTrackerCustomizer() {
      public Object addingService(ServiceReference reference) {
        Object service = context.getService(reference);
        synchronized (this) {
          if (Activator.this.airbag == null) {
            Activator.this.airbag = (IAirbag) service;
            Activator.this.bind();
          }
        }
        return service;
      }

      public void removedService(
          ServiceReference reference, Object service) {
        synchronized (this) {
          if (service != Activator.this.airbag)
            return;
          Activator.this.unbind();
          Activator.this.bind();
        }
      }

      public void modifiedService(ServiceReference reference,
          Object service) {
        // No service property modifications to handle.
      }
    };
  }

  private void bind() {
    if (gps == null) {
      gps = (IGps) gpsTracker.getService();
      if (gps == null)
        return; // No IGps service.
    }
    if (airbag == null) {
      airbag = (IAirbag) airbagTracker.getService();
      if (airbag == null)
        return; // No IAirbag service.
    }
    // Bind IGps and IAirbag to the EmergencyMonitor
    monitor.bind(gps, airbag);
  }

  private void unbind() {
    if (gps == null || airbag == null)
      return;
    monitor.unbind();
    gps = null;
    airbag = null;
  }
}
```

Declarative Services

component.xml

```
<scr:component xmlns:scr="http://www.osgi.org/xmlns/scr/v1.1.0"
    enabled="true" immediate="true“
    name="org.equinoxosgi.toast.client.emergency"
    activate="startup" deactivate="shutdown">
  <implementation class="org.equinoxosgi.toast.internal.client.emergency.EmergencyMonitor"/>
  <reference bind="setAirbag" interface="org.equinoxosgi.toast.dev.airbag.IAirbag" name="airbag"/>
  <reference bind="setGps" interface="org.equinoxosgi.toast.dev.gps.IGps" name="gps"/>
</scr:component>
```


Declarative Services

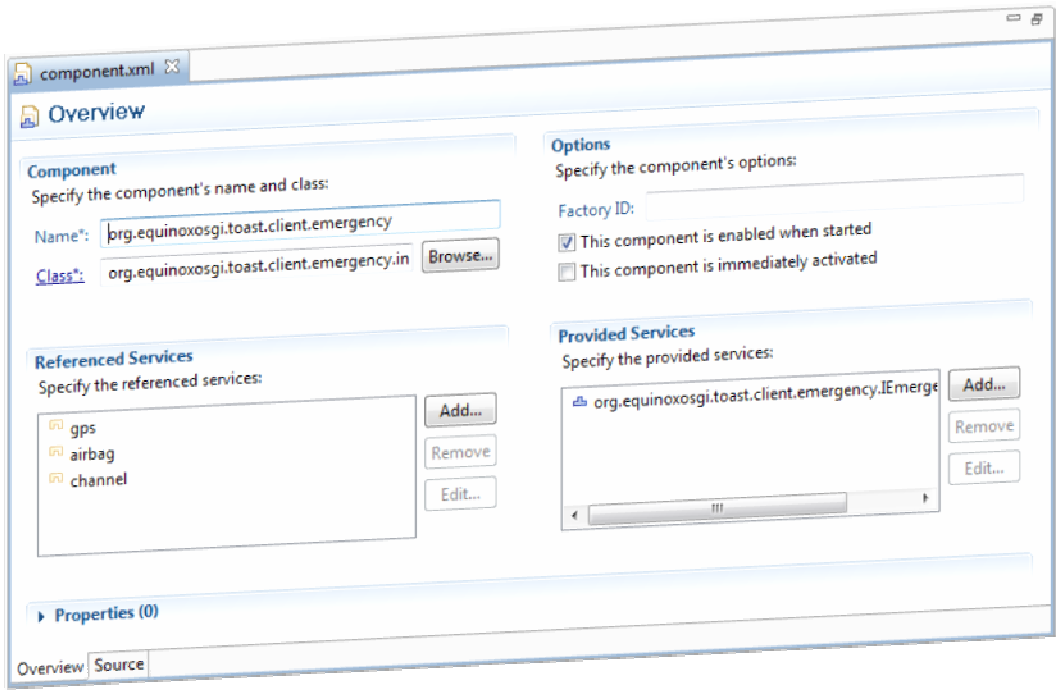
EmergencyMonitor.java

```
public class EmergencyMonitor implements IAirbagListener {
    private IAirbag airbag;
    private IGps gps;
    public void deployed() {
            System.out.println("Emergency occurred at lat=" + gps.getLatitude()
                                    + " lon=" + gps.getLongitude() + " heading=" + gps.getHeading()
                                    + " speed=" + gps.getSpeed());
    }
    public void setAirbag(IAirbag value) {
            airbag = value;
    }
    public void setGps(IGps value) {
            gps = value;
    }
    public void shutdown() {
            airbag.removeListener(this);
    }
    public void startup() {
            airbag.addListener(this);
    }
}
```


Equinox Demo

Use Declarative Services

- Encourages good componentization
- Supports lazy class loading
- Tooling in Eclipse 3.5



Summary

- Equinox provides
  - Performant, robust OSGi R4.2 framework implementation
  - Large number of high function building blocks
  - Facilities for installing, configuring and managing function
  - Various mechanisms for collaboration
- Faster and easier to create significant applications

Legal Notices

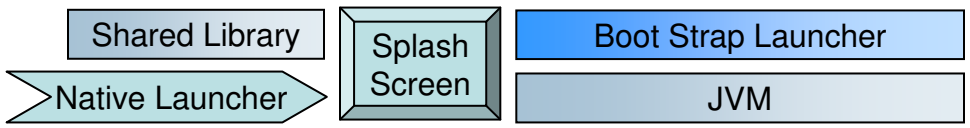
- Java and all Java-based trademarks are trademarks of Sun Microsystems, Inc. in the United States, other countries, or both
- Other company, product, or service names may be trademarks or service marks of others

Begin Backup Slides

 |

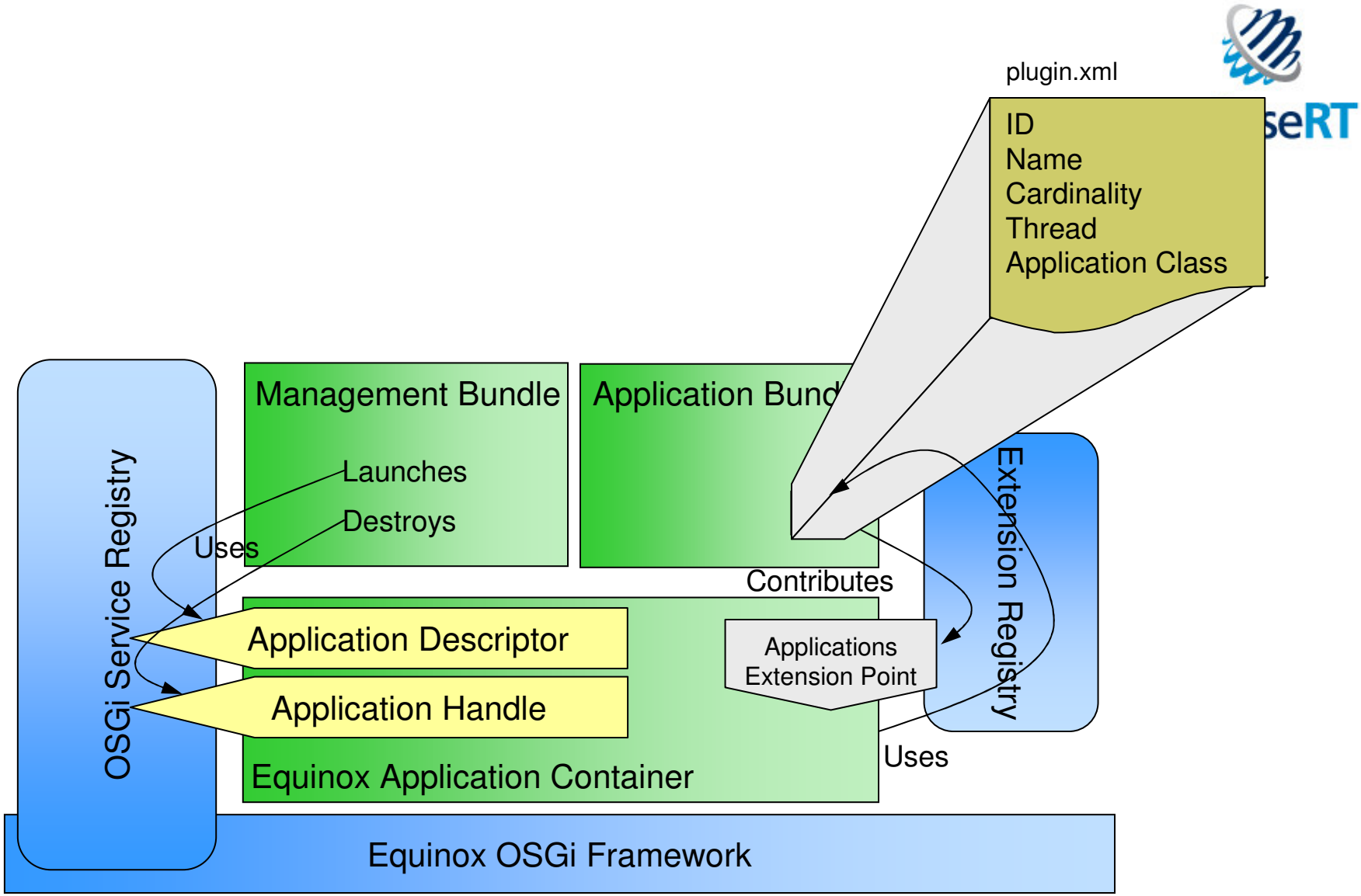
Equinox Native Launcher and Splash support

- Quick display of static splash screen
  - Displays splash before VM start
  - Splash Times: 50 ms (warm start), 400 ms (cold start)
- Uses JNI to launch the VM in-process
  - Finds and invokes the Java VM
  - Single process used instead of a separate Java process
  - Allows for SWT animation and updating of the splash during startup
- Finds the Equinox Framework and launches it
- Launcher is split between an executable, shared library and boot strap Java code
  - The shared library and boot strap Java code delivered as bundles
  - Allows for updating the executable



Equinox Application Container

- Based on OSGi Application Admin Service Specification
- Services for each application installed
  - ApplicationDescriptor – installed application
  - ApplicationHandle – running instance of installed application
- Application management
  - Manage/run multiple applications at the same time
  - Launch, Destroy, Schedule, Lock
  - Multiple agents can control applications locally, remotely, …
- Equinox allows application definition by extensions
  - Complementary use of extensions and services



 |

Equinox Security

- Enhanced signature-based **code authorization** solutions
  - Allow deployers to trade security/complexity vs. performance
  - Enforcement points include install-time (in P2), bundle load-time (**new** in Equinox), and code run-time (Java2 permissions)
- Integrated **user authentication** framework based on JAAS standard
  - Extension point based contribution of JAAS artifacts
  - Event mechanisms for monitoring login lifecycle
- New mechanisms for **user credential management**
  - User interface and service interfaces for certificate management
  - 'Secure storage' service for storing encrypted preferences

Bundle Collaboration – Extension Registry

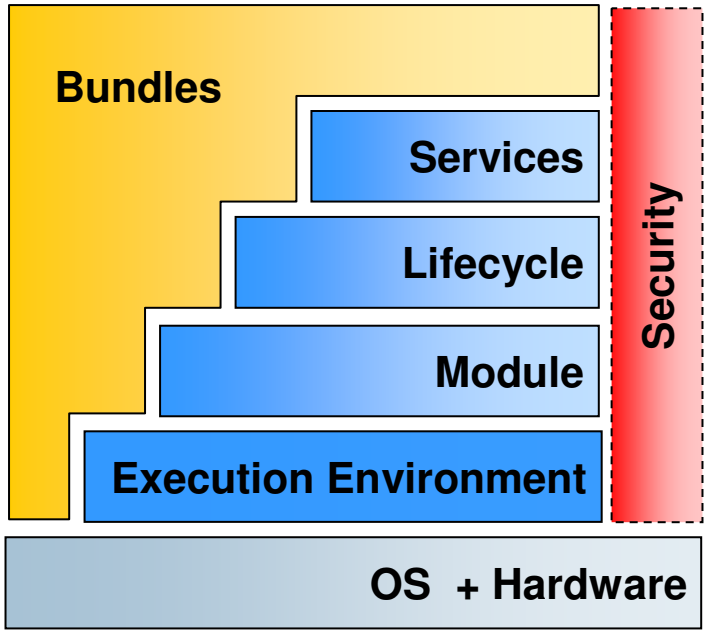
- Contract is Defined by an Extension Point
  - Contract may involve Java API, but not required
  - Additional data may be required by contract (e.g. static help content)
  - Contract is declared (in *plugin.xml*)
- Contract is implemented by Extensions
  - If required, provides a concrete implementation of Java API contract
  - May provide other data required by the contract
  - Extensions are declared (in *plugin.xml*)
- Contract consumer
  - Extension point provider consumes extensions
  - Extensions are lazily loaded as they are needed by the extension point
- Lifecycle
  - Become active when declaring bundle is in the RESOLVED state
  - Resolution state is cached to allow for quick re-launch when 1000s of them exist

Bundle Collaboration – OSGi Services

- Contract is defined by a service interface or class
  - Same service contract (package) may be shared across multiple consumers and producers of the service
  - Contract is based on Java API
- Contract is implemented by service Implementers
  - Provide a concrete implementation of service contract
  - Register the implementation object with the service registry
- Contract consumers
  - Track available services and get instances of the service through OSGi API (BundleContext)
  - The service object may be used by any bundle in the system
- Lifecycle
  - Producers and consumers of services must have a valid BundleContext
    - Bundle must be in the STARTING, ACTIVE or STOPPING states
  - Production and consumption of services is programmatic
    - Difficult to cache dependencies
- Declaration models available
  - OSGi Declarative Services
  - Spring Dynamic Modules for OSGi
  - Eclipse Service Activator Toolkit (SAT)

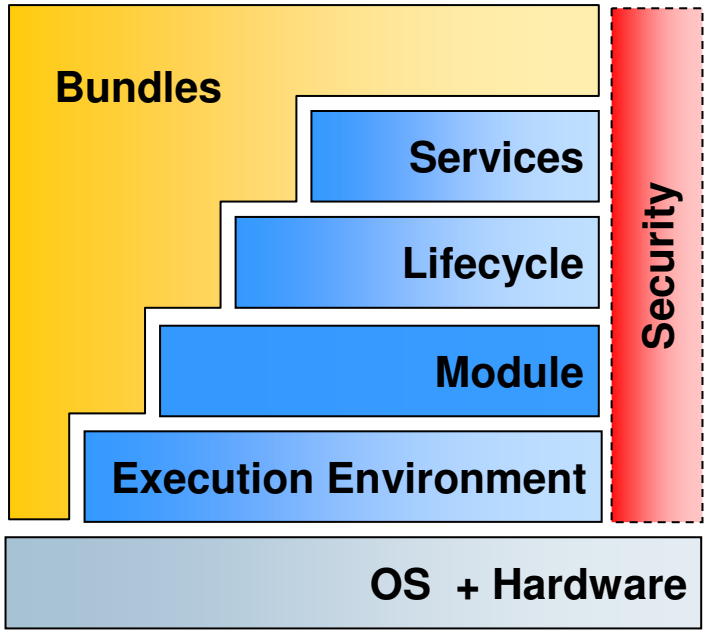
The OSGi Framework – Execution Environment

- Execution Environment
  - The VM used to launch the Framework
  - The OSGi specification originated on the J2ME platform
  - Framework implementations can scale down to small devices and scale up to large server environments



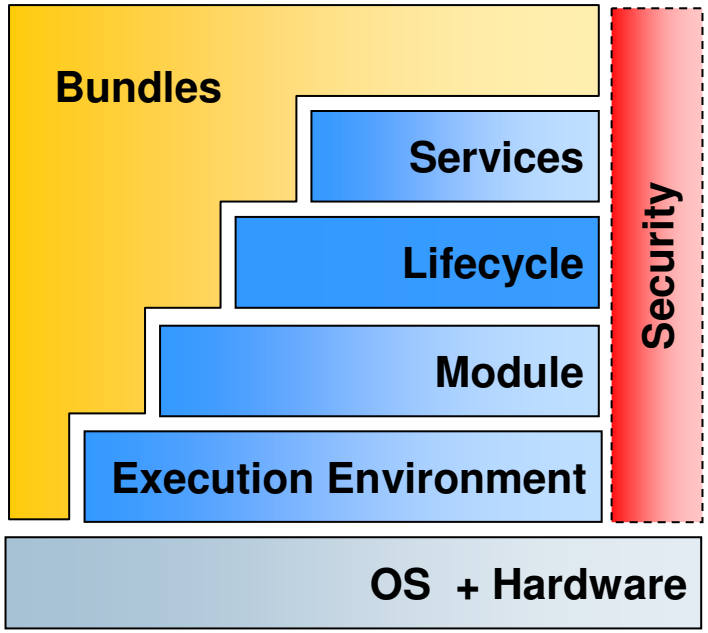
The OSGi Framework – Module Layer

- Module system for the Java Platform
  - Enforces visibility rules
  - Dependency management
  - Supports versioning of bundles, the OSGi modules
- Sophisticated modularity framework
  - provides for class space consistency for bundles
  - supports multiple versions of packages and bundles



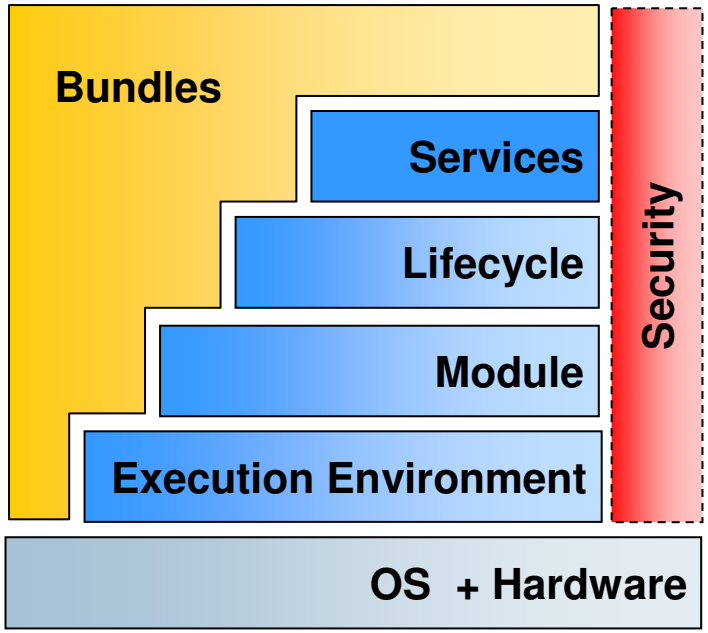
The OSGi Framework – Lifecycle Layer

- Lifecycle Layer provides API to manage bundles
  - Installing
  - Starting
  - Stopping
  - Updating
  - Uninstalling
  - All dynamically supported at runtime



The OSGi Framework – Service Layer

- Provides an in-VM service model
  - Services can be registered and consumed inside a VM
  - Again all operations are dynamic
  - Extensive support for notification of the service lifecycle



Equinox Building Blocks