What's cool in the new and updated

OSGi Specs

Carsten Ziegeler

David Bosschaert

Speakers

Carsten Ziegeler (cziegeler@apache.org)

- RnD Adobe Research Switzerland
- OSGi Board, CPEG and EEG Member
- ASF member

David Bosschaert (davidb@apache.org)

- RnD Adobe Research Dublin
- Co-chair OSGi EEG
- Open-source and cloud enthusiast

Agenda

- Framework updates
- Repository update
- Asynchronous Services & Promises
- Declarative Services
- Http Service
- Cloud
- Semantic Versioning Annotations
- Other spec updates

OSGi Core R6 Release

- **Service Scopes**
- **Package and Type Annotations**
- **Data Transfer Objects**
- Native Namespace
- WeavingHook Enhancements
- System Bundle Framework Hooks
- Extension Bundle Activators
- Framework Wiring

Framework Updates

Service Scopes (RFC 195)

OSGi R5 supports two service scopes:

- Singleton
- Bundle (ServiceFactory)

Transparent to the client

```
S getService(ServiceReference<S> ref)
void ungetService(ServiceReference<S> ref)
```

Framework Updates

Service Scopes (RFC 195)

- Third Scope: **prototype**
- Driver: Support for EEG specs (EJB, CDI)
- Usage in other spec updates

Clients need to use new API / mechanisms

Framework Updates

Service Scopes (RFC 195)

New BundleContext Methods API

```
S getServiceObjects(ServiceReference<S> ref).getService()
void getServiceObjects(ServiceReference<S> ref).ungetService(S)
```

Transparent to the client

Framework Updates

Service Scopes (RFC 195)

⦿ New **PrototypeServiceFactory** interface

⦿ Managed service registration properties for inspection

⦿ Support in component frameworks (DS, Blueprint)

Data Transfer Objects

RFC 185 – Data Transfer Objects

- Defines a DTO model for OSGi
  - Serializable/Deserializable objects
- Use cases: REST, JMX, Web Console...
- To be adopted by other specs

RFC 185 – Data Transfer Objects

Getting DTOs: adapter pattern

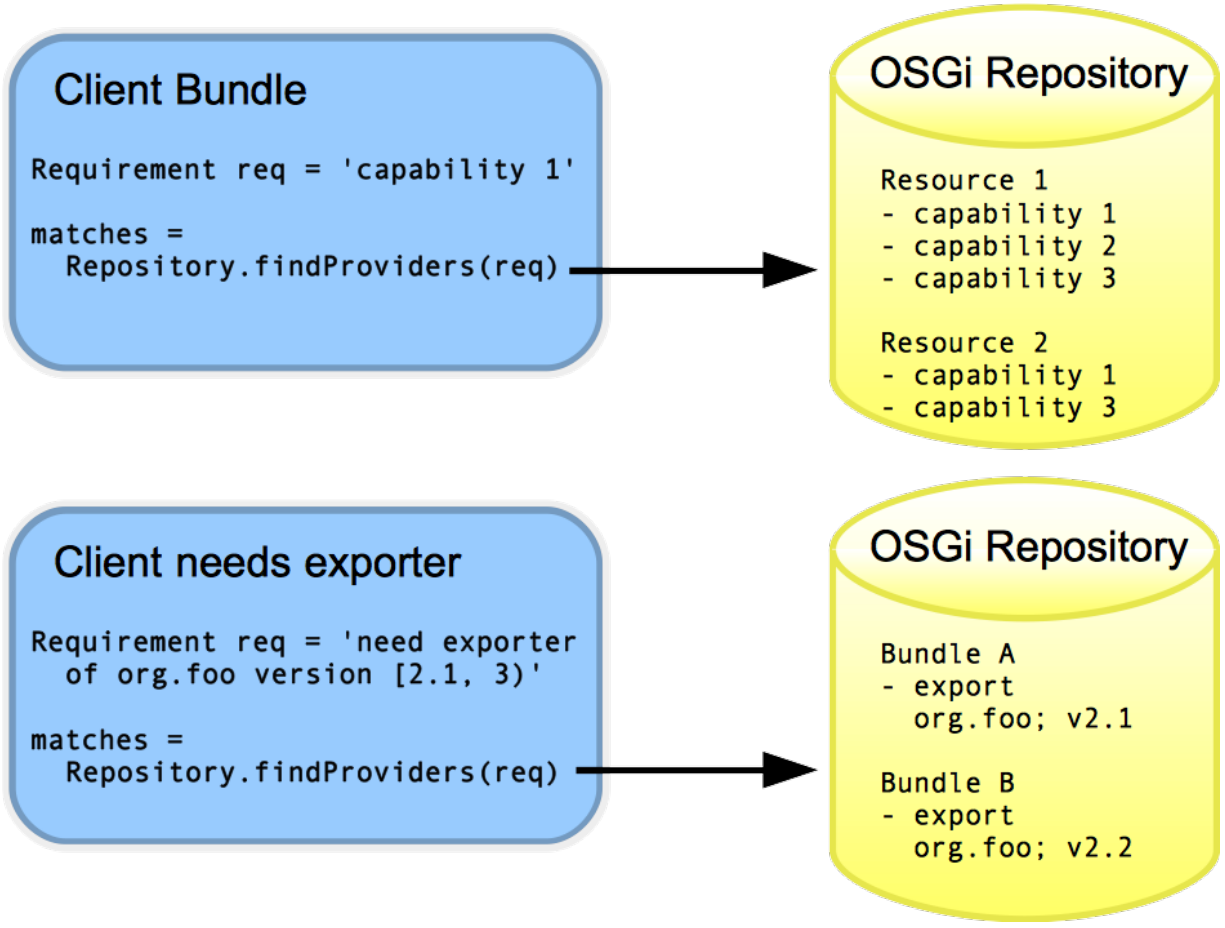
```
public class BundleDTO extends org.osgi.dto.DTO {

    public long id;

    public long lastModified;

    public int state;

    public String symbolicName;

    public String version;
}
```

RFC 185 – Data Transfer Objects

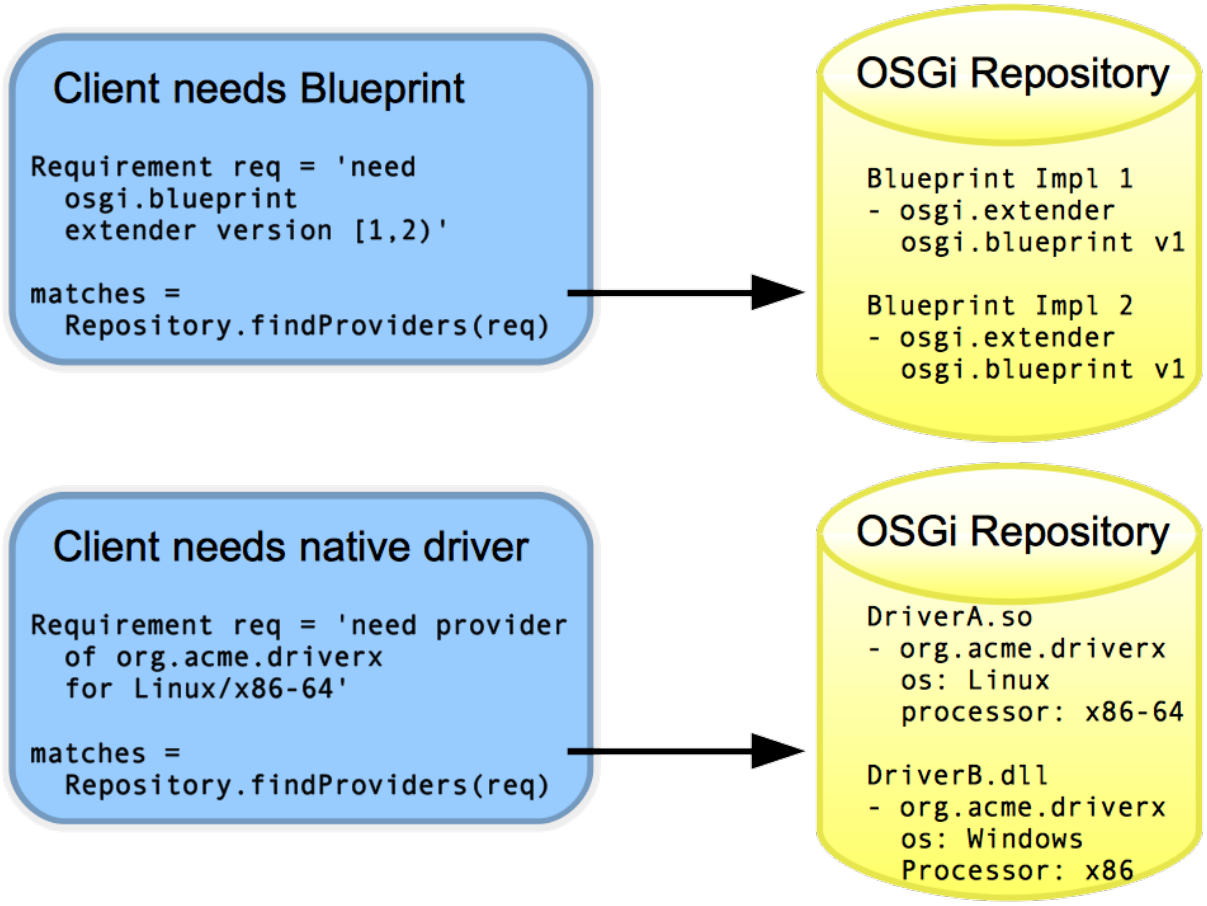
DTOs for the OSGi framework

- FrameworkDTO
- BundleDTO
- ServiceReferenceDTO
- BundleStartLevelDTO, FrameworkStartLevelD
- CapabilityDTO, RequirementDTO, ResourceD
- BundleWiringsDTO, etc

Repository 1.1



OSGi Repository today



Example Repository namespaces

RFC 187 - Repository 1.1

Existing repository powerful

but: *limited to queries in a single namespace*

New in RFC 187:

- Combine requirements spanning multiple namespaces:

```
Repository repo = ... // Obtain from Service Registry
  Collection<Resource> res = repo.findProviders(
    repo.getExpressionCombiner().and(
      repo.newRequirementBuilder("osgi.wiring.package").
        addDirective("filter","(osgi.wiring.package=foo.pkg1)").
          buildExpression(),
      repo.newRequirementBuilder("osgi.identity").
        addDirective("filter",
          "(license=http://opensource.org/licenses/Apache-2.0)").
          buildExpression()));
```

Asynchronous Services & Promises

Async Services

- Asynchronously invoke services
  - existing service
  - new ones, written for async access
- Client invokes the service via a *mediator*
- Invocation returns quickly
  - result can be obtained later

Async Services - example

A typical service:

```
public interface CalcService {
  BigInteger factorial(int num);
}
```

Invoke it asynchronously:

```
CalcService mySvc = ...  // from service registry
Async asyncService = ... // from service registry
CalcService myMediator = asyncService.mediate(mySvc);
final Promise<BigInteger> p = asyncService.call(
                           myMediator.factorial(1000000));

// ... factorial invoked asynchronously ...

// callback to handle the result when it arrives
p.onResolve(new Runnable() {
  public void run() {
    System.out.println("Found the answer: " + p.getValue());
  }
});
```

OSGi Promises

- Inspired by JavaScript Promises
  - Make latency and errors explicit
- Provide async chaining mechanism
  - Based on callbacks
- Promises are Monads
- Works with many (older) Java versions
- Designed to work with Java 8
  - CompletableFuture and Lambdas
- Used with Async Services
  - Also useful elsewhere

OSGi Promises - example

```
Success<String,String> transformResult = new Success<>() {
  public Promise<String> call(Promise<String> p) {
    return Promises.resolved(toHTML(p.getValue()));
  }
};
Promise<String> p = asyncRemoteMethod();
p.then(validateResult)
  .then(transformResult)
    .then(showResult, showError);
```

Declarative Services

RFC 190 - Declarative Services Enhancements

- Support of prototype scope
- Introspection API
- DTOs
- But most importantly...

Simplify Component Dvlpmnt

```
@Component(property={
        MyComponent.PROP_ENABLED + ":Boolean=" + MyComponent.DEFAULT_ENA
        MyComponent.PROP_TOPIC + "=" + MyComponent.DEFAULT_TOPIC_1,
        MyComponent.PROP_TOPIC + "=" + MyComponent.DEFAULT_TOPIC_2,
        "service.ranking:Integer=15"
})
public class MyComponent {

    static final String PROP_ENABLED = "enabled";
    static final String PROP_TOPIC = "topic";
    static final String PROP_USERNAME = "userName";

    static final boolean DEFAULT_ENABLED = true;
    static final String DEFAULT_TOPIC_1 = "topicA";
    static final String DEFAULT_TOPIC_2 = "topicB";
```

Simplify Component Dvlpmnt

```
@Activate
protected void activate(final Map config) {
     final boolean enabled =
        PropertiesUtil.toBoolean(config.get(PROP_ENABLED),
                                 DEFAULT_ENABLED);
     if ( enabled ) {
         this.userName =
             PropertiesUtil.toString(config.get(PROP_USERNAME), null
         this.topics =
             PropertiesUtil.toStringArray(config.get(PROP_TOPIC),
               new String[] {DEFAULT_TOPIC_1, DEFAULT_TOPIC_2});
     }
}
```

Use annotations for configuration...

```
@interface MyConfig {
    boolean enabled() default true;
    String[] topic() default {"topicA", "topicB"};
    String userName();
    int service_ranking() default 15;
}
```

...and reference them in lifecycle methods

```
@Component
public class MyComponent {

    String userName;

    String[] topics;

    @Activate
    protected void activate(final MyConfig config) {
         // note: annotation MyConfig used as interface
         if ( config.enabled() ) {
             this.userName = config.userName();
             this.topics = config.topic();
         }
    }
}
```

...or even simpler...

```
@Component
public class MyComponent {

    private MyConfig configuration;

    @Activate
    protected void activate(final MyConfig config) {
         // note: annotation MyConfig used as interface
         if ( config.enabled() ) {
            this.configuration = config;
         }
    }
}
```

Annotation Mapping

⦿ Fields registered as component properties

⦿ Name mapping (_ -> .)

⦿ Type conversion for configurations

Additional Metatype Support (RFC 208)

```
@ObjectClassDefinition(label="My Component",
                  description="Coolest component in the world.")
@interface MyConfig {
    @AttributeDefinition(label="Enabled",
           description="Topic and user name are used if enabled")
    boolean enabled() default true;

    @AttributeDefinition(...)
    String[] topic() default {"topicA", "topicB"};

    @AttributeDefinition(...)
    String userName();

    int service_ranking() default 15; // maps to service.ranking
}
```

RFC 190 - Declarative Services Enhancements

- Annotation configuration support
- Support of prototype scope
- Introspection API
- DTOs

HTTP Service

Http Whiteboard Service

RFC 189

- Whiteboard support
- Servlet API 3+ Support
- and Introspection API

Whiteboard Servlet Registration

```
@Component(service = javax.servlet.Servlet.class,
   scope="PROTOTYPE",
   property={
     "osgi.http.whiteboard.servlet.pattern=/products/*",
   })
public class MyServlet extends HttpServlet {
   ...
}
```

Whiteboard Servlet Filter Registration

```
@Component(service = javax.servlet.Filter.class,
   scope="PROTOTYPE",
   property={
     "osgi.http.whiteboard.filter.pattern=/products/*",
   })
public class MyFilter implements Filter {
   ...
}
```

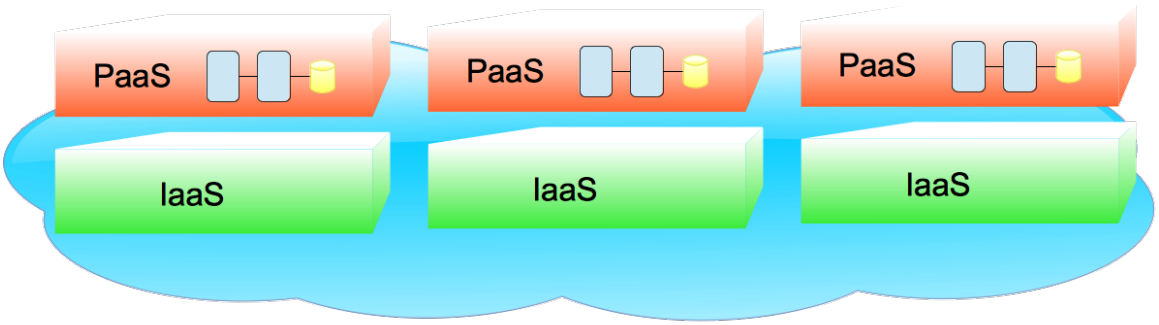
Additional Support

- Most listener types are supported

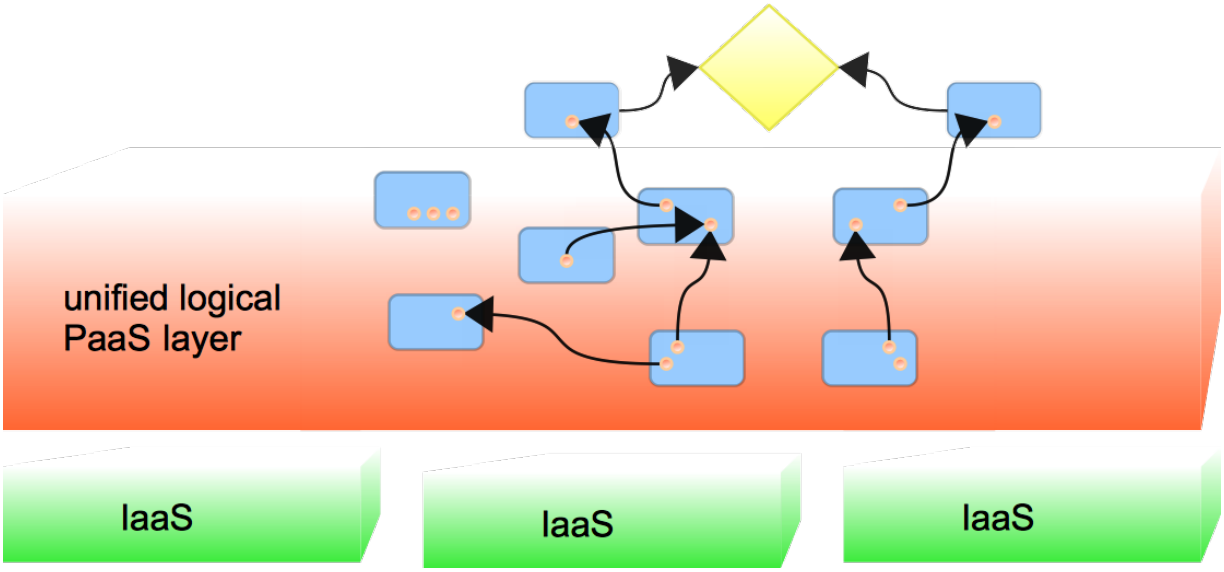
Register with their interface

- Error Pages and Resources
- Shared and segregated HttpContexts
- Target Http Service

Cloud



Current PaaS offerings...



OSGi *Cloud Ecosystems* PaaS

An OSGi cloud ecosystem...

- Many frameworks
  - hosting a variety of deployments
- Together providing *The Application*
- Not a bunch of replicas
  - rather a collection of different nodes
  - with different roles working together
  - some may be replicas
- Load varies over time
- ... and so does your cloud system
  - topology
  - configuration
  - number of nodes
  - depending on the demand

To realize this you need...

- Information!
  - need to know what nodes are available
  - ability to react to changes
- Provisioning capability
- Remote invocation
  - inside your cloud system
  - to get nodes to communicate
  - either directly...
  - ... or as a means to set up communication channels

RFC 183 - Cloud Ecosystems

`FrameworkNodeStatus` service:

- information about each Cloud node
- accessible as a Remote Service
- throughout the ecosystem

Information such as:

- Hostname/IP address
- Location (country etc)
- OSGi and Java version running
- A REST management URL

- Runtime metadata
  - Available memory / disk space
  - Load measurement

... you can add custom metadata too ...

key	*data type*	*description*
org.osgi.framework.uuid	String	The globally unique ID for this framework.
org.osgi.node.host	String+	The external host names or ip addresses for this OSGi Framework, if exists.
org.osgi.node.host.internal	String+	The internal host names or ip addresses for this OSGi Framework for access from inside the Ecosystem, if exists.
org.osgi.node.type	String	The name of the Cloud/Environment in which the Ecosystem operates.
org.osgi.node.version	String	The version of the Cloud/Environment in which the Ecosystem operates. The value follows the versioning scheme of the cloud provider and may therefore not comply with the OSGi versioning syntax.
org.osgi.node.country	String (3, optional)	ISO 3166-1 alpha-3 location where this Framework instance is running, if known.
org.osgi.node.location	String (optional)	ISO 3166-2 location where this framework instance is running, if known. This location is more detailed than the country code as it may contain province or territory.
org.osgi.node.region	String	Something smaller than a country and bigger than a location (e.g. us-east)
org.osgi.node.rest.url	String+ (URL, optional)	The external URL of the framework management REST API, if available.
org.osgi.node.rest.url.internal	String+ (URL, optional)	The ecosystem-internal URL of the framework management API, if available.
org.osgi.framework.version org.osgi.framework.processor org.osgi.framework.os.name org.osgi.framework.os. version	String	The value of the Framework properties as obtained via BundleContext.getProperty().
java.version, java.runtime.version, java.vm.vendor, java.vm.version, java.vm.name	String	The values of the corresponding Java system properties.
… additional properties …		Additional properties may appear, set by the framework, Remote Services implementation or other entity.
… custom properties …		See section 5.2.2.

`FrameworkNodeStatus` service properties

RFC 182 - REST API

A cloud-friendly remote management API works great with `FrameworkNodeStatus`

Example:

```
addingService(ServiceReference<FrameworkNodeStatus> ref) {
  // A new Node became available
  String url = ref.getProperty("org.osgi.node.rest.url");
  RestClient rc = new RestClient(new URI(url));

  // Provision the new node
  rc.installBundle(...);
  rc.startBundle(...);
}
```

Additional ideas in RFC 183

⦿ Special Remote Services config type

- `osgi.configtype.ecosystem`
- defines supported Remote Service data types
- not visible outside of cloud system

⦿ Ability to intercept remote service calls

- can provide different service for each client
- can do invocation counting (quotas, billing)

⦿ Providing remote services meta-data

- quota exceeded
- payment needed
- maintenance scheduled

Current OSGi cloud work

Provides a base line

- to build fluid cloud systems
- *portability* across clouds

Where everything is dynamic

- nodes can be repurposed

... and you deal with your cloud nodes through OSGi services

Type and Package Annotations

Semantic Versioning...

... is a versioning policy for ***exported packages***.

OSGi versions: <major>.<minor>.<micro>.<qualifier>

Updating package versions:

- fix/patch (no change to API):
  update micro
- extend API (affects implementers, not clients):
  update minor
- API breakage:
  update major

Note: not always used for *bundle* versions

RFC 197 – OSGi Type and Package Annotations

- Annotations for documenting semantic versioning information
  - Class retention annotations
- @Version
- @ProviderType
- @ConsumerType

Other Enterprise Spec updates

▻ Remote Service Admin 1.1

- Remote Service registration modification

▻ Subsystems 1.1

- Provide Deployment Manifest separately
- Many small enhancements

▻ Portable Java SE/EE Contracts

Where can I get it?

Core R6 spec released this week:

http://www.osgi.org/Specifications/HomePage

(http://www.osgi.org/Specifications/HomePage)

Enterprise R6 draft released this week:

http://www.osgi.org/Specifications/Drafts

(http://www.osgi.org/Specifications/Drafts)

RFCs 189, 190, 208 included in zip

All current RFCs at https://github.com/osgi/design

(https://github.com/osgi/design)

Questions?