RFP-151 Declarative Services Enhancements

Final
8 Pages

Abstract

10 point Arial Centered.

Declarative Services provide nice functionality to implement Dependency Injection programming in OSGi based applications. One of the goals is to limit the requirement to use OSGi specific API. This RFP proposes extensions towards this goal. In addition Declarative Services currently lacks a proper administrative API to introspect components.
0 Document Information

0.1 Table of Contents

0 Document Information ................................................................................................................. 2
  0.1 Table of Contents ...................................................................................................................... 2
  0.2 Terminology and Document Conventions ................................................................................. 2
  0.3 Revision History ...................................................................................................................... 2

1 Introduction .................................................................................................................................. 3

2 Application Domain ..................................................................................................................... 3
  2.1 Terminology + Abbreviations ................................................................................................. 4

3 Problem Description ..................................................................................................................... 4

4 Use Cases ..................................................................................................................................... 4

5 Requirements ............................................................................................................................... 4

6 Document Support ....................................................................................................................... 5
  6.1 References ............................................................................................................................... 5
  6.2 Author's Address ...................................................................................................................... 5
  6.3 End of Document ...................................................................................................................... 5

0.2 Terminology and Document Conventions

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY" and "OPTIONAL" in this document are to be interpreted as described in 6.1.

Source code is shown in this typeface.

0.3 Revision History

The last named individual in this history is currently responsible for this document.
<table>
<thead>
<tr>
<th>Revision</th>
<th>Date</th>
<th>Comments</th>
</tr>
</thead>
</table>
| Initial  | May, 7. 2012 | Initial Document listing requirements based on actual implementations in Apache Felix Declarative Services (SCR)  
Felix Meschberger, Adobe Systems, fmeschbe@adobe.com |
| Update   | June, 7. 2012 | Add more requirements from DS related requests from Member Bugzilla  
Felix Meschberger, Adobe Systems, fmeschbe@adobe.com |
| Update   | June, 27. 2012 | Update after New York F2F:  
- Remove References to Configurable (RFC-179)  
- Rephrase R5-R7  
- Remove concrete namespace from R8  
- Complete information from referenced Bugs  
Felix Meschberger, Adobe Systems, fmeschbe@adobe.com |
| Final    | Sept., 24. 2012 | Finalized to prepare for vote after Basel F2F without further changes  
Felix Meschberger, Adobe Systems, fmeschbe@adobe.com |

1 Introduction

This Declarative Services Enhancements RFP proposes to specify functionality currently implemented in some implementations of the specification or currently requiring special component code as part of the OSGi Declarative Services Specification.

2 Application Domain

Declarative Services (chapter 121 in the OSGi specifications) defines a POJO programming model for OSGi services. This model requires Service Component class be implemented in a certain way and the XML component descriptions be authored.
2.1 Terminology + Abbreviations

DS Declarative Services

POJO Plain old Java Object; term use for objects not implementing and framework specific plumbing such as Servlet API, Spring API, or OSGi API.

SCR Service Components Runtime; generally the implementation of the Declarative Services Specification; also the name of the Apache Felix implementation (Apache Felix SCR).

3 Problem Description

3.1 Management

There is no official API yet to introspect and thus manage the declared service components. To work around this missing functionality the Apache Felix project defined such an API which is also implemented by current versions of the Eclipse Equinox implementation.

This current API has some short-comings which are addressed by a new proposal.

3.2 Requirement to use OSGi API

The overall goal of DS is to limit or remove the requirement to use OSGi API in the components and thus support regular POJOs. With the latest release of the Declarative Services Specification (Version 1.2), this goal is attained to a great extent.

One thing still missing support is the ability to specify custom service registration properties. The only solution to do this in the current specification is to register the service in the activation method (and unregister in the deactivation method). But this requires use of OSGi API and also leads to non-use of a central functionality of DS, the registration of services on behalf of the declared components and thus reducing template code.

This same proposal is also asked for in Bug 2250 (allow a component to update its own service properties).

3.3 Bound Service Properties

As of DS Version 1.1 the service registration properties of bound services can be provided to the components using an optional java.lang.Map argument. While this allows for great capability introspecting the bound service it lacks support for ordering defined for org.osgi.framework.ServiceReference.

The solution applied today is to either use the greedy service binding policy option as defined in DS Version 1.2 or to implement such ordering in the component itself. Such implementation, though, is pure template code and thus error prone load to developers.
3.4 Support inheritance in the DS Annotations (Bug 2138)

The Declarative Services specification 1.2 introduced annotations to help developers with the creation of Component descriptors. The current design requires these annotations to be specified on the component implementation class. They are not processed when specified on super classes.

The main reason for this is that the annotations are processed at tooling time and not runtime. At runtime, you must have the full type hierarchy known and thus could reliably inspect super types for annotations. But at tool time (e.g. bundle packaging), you may not have the full type hierarchy available or the available type hierarchy may be different than at runtime due to package substitution. So constructing the component descriptions at tool time from information in super classes is risky.

However, there seems to be a constant stream of requests to support inheritance of DS annotations.

3.5 Create separate Service annotation (Bug 2140)

The current DS annotations define a service attribute on the @Component annotation which defaults to all directly implemented interfaces of the class.

I think it would be better to separate more between components and services and create a separate Service annotation.

If just the Component annotation is present, its not registered as a service. With the Service annotation its possible register the component as a service. The Service annotation should have a class or value attribute. If that is left empty, the component is registered for all interfaces or all directly implemented interfaces (whether all or all directly implemented might be another discussion).

Especially the default of not specifying the service attribute on the Component is error prone as it is not intuitive. Not specifying something should mean: this is no service :)

3.6 Create separate Property annotation (Bug 2141)

The current DS annotations define a property attribute on the @Component annotation which takes a key-value pair as a single string property. If a different type as string is used, this has to be coded into this string as well.

This is very error prone and forbids to use Java constants for the key or the value. In addition this notation easily clutters the @Component annotation if more than a small handful of properties is defined.

In the Apache Felix SCR annotations we have created a separate Property annotation (see http://felix.apache.org/site/scr-annotations.html#SCRAnnotations-Property) which has its own attributes for name and value like

```java
@Property(name="key", value="aString")
```

or

```java
@Property(name="anotherKey", intValue=1)
```

or

```java
@Property(name=SOME_KEY, value=SOME_VALUE)
```
This annotation can be put on a constant to derive the name:

```java
@Property(value=DEFAULT_VALUE)
public static String SOME_KEY = "key.name";
```

## 4 Use Cases

### 4.1 Management

Developers and system administrators want to introspect defined Service Components for their definition and runtime state. Currently it is not possible to introspect the Service Component Runtime other than by Java Debugging or logging which both is tedious and hardly possible in a production environment.

### 4.2 Requirement to use OSGi API

Some components require the definition of additional service registration properties or want to hide some configuration properties from service registration which cannot be turned into private configuration properties for legacy or other reasons.

### 4.3 Bound Service Properties

Sometimes components maintain a list of helper services which are consulted for some tasks in a defined order. To setup such ordering the service ranking feature defined with the `service.ranking` service registration properties comes in handy.

Currently the event methods used for service binding as defined in Section 112.3.2, Event Methods, can either take the service's `ServiceReference` or the service instance optionally along with a map of service registration properties. So the component either has to manually acquire the service object through the `ComponentContext` or the `BundleContext` or has to implement its own `Comparator` for the service registration properties map.

### 4.4 Annotation Inheritance

We have a bundle with an abstract class extended by two or more concrete implementations. The abstract class implements common functionality which requires some services. So the abstract class would add the reference annotation. The concrete implementations on the other hand rely on a protected getter method from the abstract class to access that service but it is none of their (direct) business to know how this service object gets there. They are just declared as components.

During build time the descriptors for the concrete implementations would be built where the reference annotation from the base class is taken into consideration.

```java
@Component(isAbstract = true;
public abstract class Base {
    private Service service;
    @Reference()
```
void bindService(Service service) {
    this.service = service;
}
void unbindService(Service service) {
    if (this.service == service) {
        this.service = null;
    }
}
protected Service getService() {
    return this.service;
}

@Component
public class C1 extends Base {
    ...
}
The component descriptor for C1 would then include the reference from the abstract class Base.

5 Requirements

R-1 The solution MUST define an administrative API to introspect declared components.

R-2 The solution MUST allow component instances to define their own service registration properties. The specification has to explicitly state how to deal with mandatory service registration properties (such as component.id) and private properties whose names have leading dots. See also Bug 2250 (allow a component to update its own service properties).

R-3 The solution MUST make it possible to leverage the ordering capability of the ServiceReference along with the service instance provisioning through the event method by allowing the new signature:

    void <method-name>({<parameter-type>}, ServiceReference);

R-4 The solution MUST add support for class inheritance to DS Annotations. Differences between static inheritance at build time and dynamic inheritance at runtime must be described.

R-5 The solution MUST define a new DS Annotation to declare the services exposed by a component in addition to the existing service attribute to the @Component annotation.

R-6 The solution MUST define a new DS Annotation to declare component properties for a component in addition to the existing property attribute to the @Component annotation.

R-7 The solution MUST define the osgi.extender capability for DS in accordance with the core specification rules for the osgi.extender name space.

R-8 The solution MUST support targeted PIDs according to Configuration Admin 1.5.
6 Document Support

6.1 References


6.2 Author’s Address

<table>
<thead>
<tr>
<th>Name</th>
<th>Felix Meschberger</th>
</tr>
</thead>
<tbody>
<tr>
<td>Company</td>
<td>Adobe Systems, Inc.</td>
</tr>
<tr>
<td>Address</td>
<td>Barfüsserplatz 6, 4055 Basel, Switzerland</td>
</tr>
<tr>
<td>Voice</td>
<td>+41 61 226 55 49</td>
</tr>
<tr>
<td>e-mail</td>
<td><a href="mailto:fmeschbe@adobe.com">fmeschbe@adobe.com</a></td>
</tr>
</tbody>
</table>

6.3 End of Document