RFP 146 – CDI integration

Abstract

While OSGi services are very powerful, some still find it challenging to use them effectively. This RFP looks at how CDI can be used to interact with the OSGi service layer. The intent is to bring the popular CDI programming model to OSGi as a way to interact with OSGi services. It will provide the convenience of CDI and allows developers familiar with the CDI technology to reuse their skills in an OSGi context.
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 .............................................................................................................. 3

1 Introduction ......................................................................................................................... 4

2 Application Domain ............................................................................................................. 4
  2.1 CDI .................................................................................................................................. 5
    2.1.1 Example .................................................................................................................. 5
  2.2 Weld-OSGi ..................................................................................................................... 6
    2.2.1 Weld-OSGi example ................................................................................................. 6
  2.3 Declarative Services, Blueprint and CDI .......................................................................... 6
  2.4 Terminology + Abbreviations ......................................................................................... 7

3 Problem Description ............................................................................................................ 7

4 Use Cases ............................................................................................................................. 7
  4.1 Use of CDI as a programming model in OSGi ................................................................. 7
  4.2 Inject OSGi Services into JavaEE CDI beans .................................................................. 8
  4.3 Using CDI beans through the OSGi Service Registry .................................................... 8

5 Requirements ......................................................................................................................... 8
  5.1 Functional Requirements ............................................................................................... 8
  5.2 Non-functional Requirements ......................................................................................... 10
  5.3 Requirements from RFP 98 (OSGi/Java EE umbrella RFP) ........................................ 10

6 Document Support ............................................................................................................... 10
  6.1 References ....................................................................................................................... 10
  6.2 Author’s Address ............................................................................................................. 11
  6.3 End of Document ............................................................................................................. 11

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>
<tbody>
<tr>
<td>0.1</td>
<td>November, 2011</td>
<td>Pete Muir (Red Hat), David Bosschaert (Red Hat) – initial version.</td>
</tr>
<tr>
<td>0.2</td>
<td>November, 2011</td>
<td>Incorporating feedback from London F2F.</td>
</tr>
<tr>
<td>0.3</td>
<td>April, 2012</td>
<td>Prepare for Paris F2F.</td>
</tr>
<tr>
<td>0.4</td>
<td>May, 2012</td>
<td>David Bosschaert (Red Hat), Martin Kouba (Red Hat) and Graham Charters (IBM) – Process feedback from Paris F2F</td>
</tr>
<tr>
<td>0.5</td>
<td>May, 2012</td>
<td>David Bosschaert (Red Hat), Martin Kouba (Red Hat), process feedback from the EEG concall May 9th.</td>
</tr>
<tr>
<td>0.6</td>
<td>June, 2012</td>
<td>David Bosschaert (Red Hat), Martin Kouba (Red Hat), process feedback received via <a href="http://www.osgi.org/bugzilla/show_bug.cgi?id=141">http://www.osgi.org/bugzilla/show_bug.cgi?id=141</a></td>
</tr>
<tr>
<td>0.7</td>
<td>June, 2012</td>
<td>David Bosschaert (Red Hat), feedback received during the NYC F2F.</td>
</tr>
<tr>
<td>0.8</td>
<td>June, 2012</td>
<td>David Bosschaert (Red Hat), incorporate feedback received from BJ Hargrave (IBM) and via bug 141.</td>
</tr>
<tr>
<td>0.9</td>
<td>June, 2012</td>
<td>David Bosschaert (Red Hat), incorporate feedback from the EEG call June 27th.</td>
</tr>
<tr>
<td>1.0</td>
<td>July, 2012</td>
<td>David Bosschaert (Red Hat), incorporate feedback from JJ Snyder (Oracle), prepare for vote</td>
</tr>
<tr>
<td>1.1</td>
<td>July, 2012</td>
<td>Incorporate further feedback received via bug 141.</td>
</tr>
<tr>
<td>1.2</td>
<td>Aug, 2012</td>
<td>Graham Charters (IBM), incorporated feedback from Joe Bergmark (IBM).</td>
</tr>
<tr>
<td>1.3</td>
<td>Aug, 2012</td>
<td>David Bosschaert (Red Hat), incorporate feedback from EEG call July 25th.</td>
</tr>
<tr>
<td>1.4</td>
<td>Aug, 2012</td>
<td>David Bosschaert (Red Hat), incorporate feedback from EEG call August 29th.</td>
</tr>
<tr>
<td>1.5</td>
<td>Sep, 2012</td>
<td>Mike Keith (Oracle), feedback during the EEG F2F Basel.</td>
</tr>
<tr>
<td>1.6</td>
<td>Oct, 2012</td>
<td>Minor change suggested by JJ Snyder (Oracle)</td>
</tr>
</tbody>
</table>
1 Introduction

While OSGi services are very powerful, consuming them has been a challenge for many OSGi users. There have been a number of solutions to this problem both in OSGi specifications as well as in non-standardized technologies. OSGi Declarative Services and Blueprint are popular specifications in this area, however they provide new programming models that users need to learn. As of JavaEE 6, CDI (JSR 299) is included as a standard injection technology for JavaEE components. The CDI programming model seems suitable for interaction with the OSGi service layer as well and has the benefit that developers who are familiar with CDI don't need to learn a new technology in order to interact with the OSGi service registry.

This document proposes that OSGi will support CDI with the goal of creating a specification that describes how the CDI programming model can be used to interact with OSGi services.

2 Application Domain

Software developers often need to build loosely coupled applications. The need for this stems from a number of factors:

- Developing reusable services for consumption outside of the team
- Allowing those services to be easily consumed
- Unit testing of applications and services
- Allowing larger teams to work effectively together by isolating areas of development

Software developers also wish to using a standardized programming model. This promotes:

- Transferability of skill sets
- Ease of sourcing new developers and low initial overhead
- Clear understanding of correct behavior when unexpected behavior is encountered
- Consistency of programming model across the technological strata to provide a uniformity of approach to aid understanding

Finally, software developers require an environment in which the focus can be on solving business issues rather than technological issues. This allows a more responsive development process.
2.1 CDI

CDI, Contexts and Dependency Injection is specified by JSR 299. It defines a clean, mostly annotations-based injection model which has recently become very popular. CDI is part of JavaEE 6 but can also be used standalone in a JavaSE context.

Weld ([http://seamframework.org/Weld](http://seamframework.org/Weld)) is the Reference Implementation of JSR 299.

2.1.1 Example

Although many advanced features are available, the most basic annotation used in CDI is `javax.inject.Inject` which declares the injection points for CDI.

For example the following Servlet class uses CDI injection to obtain an implementation of the WeatherBean interface.

```java
public class CDIServlet extends HttpServlet {
    @Inject
    WeatherBean weatherBean;

    @Override
    protected void doGet(HttpServletRequest req, HttpServletResponse resp)
    throws ServletException, IOException {
        PrintWriter writer = resp.getWriter();
        writer.print("The Weather in Amsterdam: "+
            weatherBean.getDescription("Amsterdam")).
            writer.flush();
        writer.close();
    }
}
```

While for the most basic use a CDI provider does not need to be annotated, CDI will attempt to find an implementor class and instantiate it using a no-arg constructor. Other mechanisms to publish a bean into CDI can be defined by using the `javax.enterprise.inject.Produces` annotation. Additionally, a number of scopes are defined that can be used to the declare the lifecycle of a CDI bean.

For example, the WeatherBean above can be scoped to the application lifecycle by adding the `javax.enterprise.context.ApplicationScoped` annotation, as in this example:

```java
public class WeatherBeanProducer {
    @Produces @ApplicationScoped
    public WeatherBean newWeatherBean() {
        WeatherBean wb = new WeatherBeanImpl();
        wb.initialize();
        return wb;
    }

    public void disposeWeatherBean(@Disposes WeatherBean wb) {
        wb.cleanup();
    }
}
```

For more information see the CDI specification at JSR 299 [3].
2.2 Weld-OSGi

The Weld-OSGi project (http://mathieuancelin.github.com/weld-osgi/) has created an integration between CDI and OSGi. It allows CDI beans to be exposed as OSGi services and CDI injections to be satisfied by OSGi services. Weld-OSGi takes additional OSGi features into account such as service registration properties and the dynamic aspects of the Service Registry.

Furthermore, Weld-OSGi provides annotation based injection for the Bundle, BundleContext, Bundle Headers and the private bundle storage facility.

Additionally Weld-OSGi provides annotations-based integration with Service and Bundle events.

2.2.1 Weld-OSGi example

Many examples can be found in the weld-osgi documentation [4].

Weld-OSGi typically uses additional annotations to interact with the OSGi service Registry. For example, the org.osgi.cdi.api.extension.annotation.Publish annotation publishes the CDI bean in the OSGi Service Registry:

```java
@Publish
@ApplicationScoped
public class MyServiceImpl implements MyService {
    @Override
    public void doSomething() { ... }
}
```

To have a CDI injection come from the OSGi Service Registry, use the OSGiService annotation:

```java
@Inject @OSGiService MyService service;
```

OSGi Services can also be selected by using LDAP filters:

```java
@Inject @OSGiService @Filter("&\(lang=EN\)\(country=US\)\") MyService service;
```

For more examples, see the weld-osgi documentation.

2.3 Declarative Services, Blueprint and CDI

In Java EE, the EJB and CDI containers are able to collaborate such that EJB manages an EJB component's lifecycle, whilst CDI manages its runtime dependencies. For example, when a new EJB is created it can be handed over to the CDI container for it to process the injections (@Inject) before finally being made available for use. This relationship helps ensure a complementary positioning between the different component models and reduces runtime duplication (EJB is not required to handle @Inject processing itself).

OSGi has two existing component models in the form of Declarative Services and Blueprint. Each has its own mechanism for injection of services and Blueprint also supports bean injection within a bundle. Neither has standards support for runtime annotations for injection, although there is some Blueprint prototype work in Apache Aries. In addressing any requirements for runtime annotations support, serious consideration should be given to the use of existing annotations, such as @Inject. It also makes sense to consider creating similar complementary relationship between their containers and the CDI container for runtime injection processing, thus reducing duplication between various component model containers.
2.4 Terminology + Abbreviations

CDI – Context and Dependency Injection for JavaEE. Specified in JSR 299.

3 Problem Description

CDI provides a standardized, type-safe, loosely coupled programming model for Java EE 6 and above. Furthermore, it introduces powerful extensibility into the Java EE programming model, and promotes an ecosystem of “portable extensions”.

CDI is declarative, with metadata provided via annotations. This allows developers to locate all logic and metadata in a single location, allowing easier comprehension of the application.

CDI does not specify any modularity or inter-application communication, relying instead on the Java EE platform to provide this.

OSGi provides the de facto standard within Java for modular, service orientated programming.

Use of CDI in the context of OSGi provides a very compelling programming model. However, today there is no standard way to achieve this. A standard for leveraging CDI in OSGi will provide a migration path between JavaEE and OSGi where developers familiar with CDI can reuse their skills in both contexts without being locked in to a particular implementation.

4 Use Cases

4.1 Use of CDI as a programming model in OSGi

A software developer familiar with the CDI programming model needs to develop OSGi bundles. Rather than learning new programming models associated with the use of OSGi services, the developer wants to continue using CDI as a way to create and consume components. This will leverage the existing CDI skills in the company. There is already a project available that can do this, but the developer has strong opinions about software standards and will not use proprietary technologies, therefore he needs a standardized way to use CDI within an OSGi environment.
4.2 Inject OSGi Services into JavaEE CDI beans

Company ACME has a large number of JavaEE deployments, most of which are implemented using CDI beans. The company has made an architectural decision to start moving towards modular development based on OSGi, but this needs to be done in a gradual manner. One way to enable this is to support that CDI bean injections are coming from the OSGi Service Registry so that existing JavaEE components can be gradually replaced by OSGi services.

4.3 Using CDI beans through the OSGi Service Registry

Company ACME, which is moving towards OSGi-based development needs to be able to build on top of its existing assets many of which are represented as JavaEE CDI beans. As ACME is moving towards an OSGi services based model they need to be able to both consume other OSGi services as well as existing CDI beans through the OSGi Service Registry in order to form a unified development experience.

5 Requirements

5.1 Functional Requirements

CDI001 – The specification MUST make it possible to use the CDI annotations and XML descriptor in an OSGi bundle to expose and consume CDI beans.

CDI002 – The specification MUST make it possible to access all CDI managed beans from the OSGi Service Registry.

CDI003 – The specification MUST make it possible to consume OSGi services in CDI @Inject injection points in an OSGi bundle.

CDI004 – The specification MUST make it possible to select OSGi services used in CDI beans based on OSGi filters.

CDI005 – The specification MUST make it possible to consider CDI qualifiers when looking up CDI beans in the OSGi Service Registry.

CDI014 – The specification MUST provide a mechanism to specify additional OSGi service registration properties for CDI beans.

CDI006 – The specification MUST make it possible to write a portable CDI jar that runs both in JavaEE as well as in OSGi.

CDI007 – The specification MUST consider the thread-safety issues that can arise when migrating CDI beans from JavaEE to OSGi.

CDI008 – The specification MUST consider the issues that can arise in relation to the dynamic bundle lifecycle in OSGi.
CDI015 – The specification MUST consider the issues that can arise with OSGi service dynamism when these services are injected into a CDI bean.

CDI009 – The specification MUST make it possible to take advantage of the dynamic service capabilities of OSGi.

CDI016 – The specification MUST extend the life-cycle dependency model as provided in CDI, to support the dynamic life-cycle provided by OSGi. For example, it MUST NOT be fatal to deploy a CDI bean that does not have all its dependencies initially satisfied and it MUST be possible to change bean dependencies without requiring the CDI application to be redeployed or restarted.

CDI031 – The specification MUST extend the life-cycle dependency model of CDI to include dynamic OSGi service dependencies.

CDI017 – The specification SHOULD make it possible to declare a CDI injection point as optional.

CDI018 – The specification MUST provide a mechanism to consume multiple matching services/beans of a given type in an injection point. For example via the @Inject Instance<T> mechanism.

CDI019 – The specification MUST support CDI events as defined by the CDI specification.

CDI021 – The specification MAY provide a deep integration between CDI events and OSGi events or other OSGi mechanism.

CDI020 – The specification MUST support CDI extensions as defined by the CDI specification.

CDI022 – The specification MAY provide a deep integration between CDI extensions and OSGi services or other OSGi mechanism.

CDI010 – The specification MAY introduce additional annotations.

CDI011 – The specification MUST define the behavior in case of incorrect CDI metadata.

CDI012 – The specification MUST NOT prevent the use of @Inject (and other common java annotations) in other component models/technologies present in the OSGi Framework.

CDI013 – The specification MUST define an opt-in mechanism. Bundles not opting in MUST not be considered by the CDI-OSGi integration layer.

CDI023 – All the inter-bundle interaction between CDI beans MUST go through the OSGi Service Registry.

CDI024 – The specification MUST make it possible to access the BundleContext from inside a CDI bean in an OSGi Framework.

CDI025 – The specification SHOULD provide activation and de-activation callbacks similar to the BundleActivator methods to CDI beans in an OSGi Framework.

CDI026 – The specification SHOULD consider defining behavior for relevant CDI scopes.

CDI027 – The solution MAY define new scopes for use with CDI inside an OSGi Framework.

CDI028 – The specification MUST define an opt-in mechanism for CDI extensions.
CDI029 – The specification MUST consider the issues that arise from dynamically adding CDI extensions to the system.

CDI030 – The specification MUST support the inclusion of CDI beans and descriptors in a Web Application Bundle in the same way they can be included in a WAR (e.g. including beans.xml in WEB-INF/).

CDI032 – The specification MUST support the OSGi Service Permission security model when publishing OSGi services from CDI beans and injecting services into CDI beans. It needs to take into account that the CDI extender acts on behalf of other bundles and uses the permissions associated with those.

5.2 Non-functional Requirements

CDI050 – The specification MUST NOT prevent an implementation from injecting OSGi services into CDI beans which are not deployed as OSGi bundles.

CDI052 – The specification MUST NOT prevent and implementation from CDI050 – The specification MUST NOT prevent an implementation from injecting OSGi services into CDI beans which are not deployed as OSGi bundles.

CDI051 – The specification SHOULD adhere to the current CDI programming model as much as possible.

5.3 Requirements from RFP 98 (OSGi/Java EE umbrella RFP)

JEE001 – A Java EE/OSGi system SHOULD enable the standard Java EE application artifacts (e.g. web application) to remain installed when a supporting Java EE runtime element (e.g. web container) is dynamically replaced.

JEE002 – RFCs that refer to one or more Java EE technologies MUST NOT impede the ability of an OSGi-compliant implementation to also be compliant with the Java EE specification.

JEE003 – RFCs that refer to one or more Java EE technologies MAY define the additional aspects of the technology that are required for the technology to be properly integrated in an OSGi framework but MUST NOT make any syntactic changes to the Java interfaces defined by those Java EE specifications.

JEE004 – RFCs whose primary purpose is integration with Java EE technologies MUST NOT require an OSGi Execution Environment greater than that which satisfies only the signatures of those Java EE technologies.

6 Document Support

6.1 References


### 6.2 Author’s Address

<table>
<thead>
<tr>
<th>Name</th>
<th>Pete Muir</th>
</tr>
</thead>
<tbody>
<tr>
<td>Company</td>
<td>Red Hat</td>
</tr>
<tr>
<td>Address</td>
<td></td>
</tr>
<tr>
<td>Voice</td>
<td></td>
</tr>
<tr>
<td>e-mail</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Name</th>
<th>David Bosschaert</th>
</tr>
</thead>
<tbody>
<tr>
<td>Company</td>
<td>Red Hat</td>
</tr>
<tr>
<td>Address</td>
<td></td>
</tr>
<tr>
<td>Voice</td>
<td></td>
</tr>
<tr>
<td>e-mail</td>
<td><a href="mailto:david@redhat.com">david@redhat.com</a></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Name</th>
<th>Martin Kouba</th>
</tr>
</thead>
<tbody>
<tr>
<td>Company</td>
<td>Red Hat</td>
</tr>
<tr>
<td>Address</td>
<td></td>
</tr>
<tr>
<td>Voice</td>
<td></td>
</tr>
<tr>
<td>e-mail</td>
<td><a href="mailto:mkouba@redhat.com">mkouba@redhat.com</a></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Name</th>
<th>Graham Charters</th>
</tr>
</thead>
<tbody>
<tr>
<td>Company</td>
<td>IBM</td>
</tr>
<tr>
<td>Address</td>
<td></td>
</tr>
<tr>
<td>Voice</td>
<td></td>
</tr>
<tr>
<td>e-mail</td>
<td></td>
</tr>
</tbody>
</table>

### 6.3 End of Document