A Formal Model of the OSGi Web Container (0.1)

Rob Harrop

May 6, 2009

We model aspects of OSGi Web Container (RFC66).

Contents

1 Introduction

This specification describes the OSGi Web Container service. The Web Container service extends an OSGi service platform to support the installation of WAR files adhering to the Servlet 2.5 specification.

The Web Container provides no new installation artefacts for the OSGi service platform. Instead, WAR files are installed into the service platform as standard OSGi bundles.

The Web Container provides an *extender* that recognises bundles that are also valid WAR files. We refer to these bundles as *WAR bundles*. The extender reacts to `STARTED` and `STOPPED` events on these WAR bundles and starts and stops the corresponding web application appropriately.

The Web Container provides two main installation modes. The first mode allows a WAR bundle to be installed directly into the Web Container. In this mode, the Web Container performs no transformations to the bundle during or after installation.

The second mode allows for WAR files to be transformed during installation, with the container ensuring that result is a valid WAR bundle. The transformation process is complex and considers input from three sources. It should be noted that the WAR file going through the transformation may already be a valid WAR bundle. This specification aims to capture the exact semantics of the transformations being performed.

References

[1] OSGi Service Platform Core Specification v4.1, April 2007
`http://www.OSGi.org`

[2] Servlet 2.5 Specification
`http://jcp.org/aboutJava/communityprocess/mrel/jsr154/index2.html`

[3] The OSGi Extender Model
`http://www.osgi.org/blog/2007/02/osgi-extender-model.html`

2 Basic Types

We are interested in various identifiers, names, versions and attributes of both bundles and WARs.

2.1 Identifying Bundles

Bundles are uniquely identified by their symbolic name and version

$$[SymbolicName, Version]$$

We are interested in the ordering of Versions. There is an earliest version number which we can name here. All other versions are later than this one.

$$EARLIEST\_VERSION : Version$$

We define a relation between all Versions and the Versions that are greater than or equal to those Versions

$$\begin{array}{|l} vgeq : Version \leftrightarrow Version \\ \hline \forall\, v : \mathrm{dom}\, vgeq \bullet v \mapsto v \in vgeq \\ \forall\, v1, v2 : \mathrm{dom}\, vgeq \mid v1 \neq v2 \bullet v1 \mapsto v2 \in vgeq \Rightarrow v2 \mapsto v1 \notin vgeq \\ \neg\, (\exists\, v : \mathrm{dom}\, vgeq \mid v \neq EARLIEST\_VERSION \\ \qquad \bullet\, v \mapsto EARLIEST\_VERSION \in vgeq) \end{array}$$

2.2 Bundle Manifest Versions

Bundles have a `Bundle-ManifestVersion` that controls the behaviour applied to the bundle by the OSGi Service Platform. `Bundle-ManifestVersions` are simply numbers:

$$BundleManifestVersion == \mathbb{N}$$

2.3 Bundle Class Path

Each bundle has a private class path. The class path is made up of class path entries:

$$[BundleClassPathEntry]$$

2.4 Imports and Exports

Bundles may import and export zero or more packages. An imported package is quite different from an exported package

$[PackageName]$

$$\begin{array}{|l} \textit{ImportedPackage} \\ \hline name : PackageName \\ version : Version \\ \hline \end{array}$$

$[ExportedPackage]$

All WAR files and bundles have a file name associated with them

$[FileName]$

3 Transformation

The transformation process takes a WAR file and generates a valid bundle.

Transformations to a given property in the WAR generally operate on the value of the property in the WAR, an optional user value for this property and a default value for this property.

3.1 Bundle Symbolic Name Transformation

For the purpose of this specification we only allow bundles that comply with OSGi R4.1 or above. Therefore, all bundles must have a symbolic name

BundleSN

$$bsn : SymbolicName$$

WAR files optionally have a symbolic name

WarSN

$$wsn : Opt\ SymbolicName$$

A mechanism exists for translating the file name of the WAR into a suitable symbolic name.

$$defaultSymbolicName : FileName \rightarrow SymbolicName$$

WAR symbolic name are transformed into Bundle symbolic names according to the following rules:

- if the user supplies a symbolic name it is used; otherwise
- if the WAR already has a symbolic name it is used; otherwise
- a symbolic name is generated from the file name

TransformSN

$$WarSN$$
$$BundleSN'$$
$$usn? : Opt\ SymbolicName$$
$$fn? : FileName$$

$$bsn' = usn?\ \mathsf{defaultsTo}\ (wsn\ \mathsf{defaultsTo}\ (defaultSymbolicName\ fn?))$$

3.2 Bundle Version Transformation

Bundles must have a version

$$\begin{array}{|l} \textit{BundleVersion} \\ \hline bv : \textit{Version} \\ \hline \end{array}$$

WARs may optionally have a version

$$\begin{array}{|l} \textit{WarVersion} \\ \hline wv : \textit{Opt Version} \\ \hline \end{array}$$

WAR versions are transformed into Bundle versions according to the following rules:

- if the user supplies a version it is used; otherwise
- if the WAR already has a version it is used; otherwise
- the default `EARLIEST_VERSION` is used

$$\begin{array}{|l} \textit{TransformVersion} \\ \hline \textit{WarVersion} \\ \textit{BundleVersion}' \\ uv? : \textit{Opt Version} \\ \hline bv' = uv? \ \mathsf{defaultsTo}\ (wv \ \mathsf{defaultsTo}\ \textit{EARLIEST\_VERSION}) \\ \hline \end{array}$$

3.3 Bundle Manifest Version Transformation

Bundles must have a bundle manifest version of at least 2

$$\begin{array}{|l} \textit{BundleMV} \\ \hline bmv : \textit{BundleManifestVersion} \\ \hline bmv \geq 2 \\ \hline \end{array}$$

WARs may optionally have a manifest version. There is no restriction on the value of a WARs bundle manifest version

WarMV

$$wmv : Opt\ BundleManifestVersion$$

WAR bundle manifest versions are transformed into bundle manifest versions according to the following rules

- if the user supplies a bundle manifest version greater than 2 it will be used; otherwise
- if the WAR contains a bundle manifest version greater than 2 it will be used; otherwise
- the bundle manifest version is set to 2

You should note that the bundle manifest version must always be at least 2.

TransformMV

$$WarMV$$
$$BundleMV'$$
$$umv? : Opt\ BundleManifestVersion$$

$$\textbf{let}\ mv == umv?\ \mathsf{defaultsTo}\ (wmv\ \mathsf{defaultsTo}\ 2) \bullet$$
$$mv \geq 2 \Rightarrow bmv' = mv \land mv < 2 \Rightarrow bmv' = 2$$

3.4 Bundle Class Path Transformation

All bundles have a class path with at least one entry. The bundle class path cannot have duplicate entries.

BundleClassPath

$$bcp : \mathrm{iseq}\ BundleClassPathEntry$$

$$\#bcp > 0$$

WARs need not have bundle class path entries already, and they may have duplicates entries.

WarClassPath

$$wcp : \mathrm{seq}\ BundleClassPathEntry$$

The transformation process of the WAR class path must transform the bundle class path to meet the following constraints:

- all unique entries in the class path of the WAR must be present in the transformed class path
- `WEB-INF/classes` must be the first entry in the transformed class path
- all library JARs in `WEB-INF/lib` must have a corresponding in the transformed class path

We need a way to represent the `WEB-INF/classes` bundle class path entry

$$WEB\_INF\_CLASSES : BundleClassPathEntry$$

We also need a way to represent library JARs

$$[Lib]$$

A library JAR can be converted into a class path entry

$$libToClassPath : Lib \rightarrow BundleClassPathEntry$$

Thus the bundle class path transformation is defined as

TransformBundleClassPath

$$WarClassPath$$
$$BundleClassPath'$$
$$libs? : \mathbb{P}\, Lib$$

$$\operatorname{ran} wcp \subseteq \operatorname{ran} bcp'$$
$$libToClassPath(\!|\, libs? \,|\!) \subseteq \operatorname{ran} bcp'$$
$$head\ bcp' = WEB\_INF\_CLASSES$$

3.5 Import Package Transformation

A bundle can import any number of packages. It may only import a package with a given name once

BundleImports

$$bi : \mathbb{P}\, ImportedPackage$$

$$\forall p1, p2 : bi \mid p1.name = p2.name \bullet p1 = p2$$

A WAR may import any number of packages. There are no restrictions on the packages imported

WarImports

$wi : \mathbb{P}\, ImportedPackage$

TransformImports

$WarImports$

$BundleImports'$

$ui? : \mathbb{P}\, ImportedPackage$

3.6 Full Transformation

$$Bundle \mathrel{\widehat=} BundleSN \land BundleVersion \land BundleMV \land BundleClassPath$$

$$War \mathrel{\widehat=} WarSN \land WarVersion \land WarMV \land WarClassPath$$

$$TransformWarToBundle \mathrel{\widehat=} TransformSN \land TransformVersion \land TransformMV \land TransformBundleClassPath$$