OSGi Features

Draft

14 Pages

Abstract

OSGi is regularly used as a platform for running applications comprised of a large number of bundles, configurations and other artifacts. However it is lacking a developer friendly mechanism to define such applications. The requirements in this RFP aim at providing a solution to this.

0 Document Information

0.1 License

DISTRIBUTION AND FEEDBACK LICENSE, Version 2.0

The OSGi Alliance hereby grants you a limited copyright license to copy and display this document (the “Distribution”) in any medium without fee or royalty. This Distribution license is exclusively for the purpose of reviewing and providing feedback to the OSGi Alliance. You agree not to modify the Distribution in any way and further agree to not participate in any way in the making of derivative works thereof, other than as a necessary result of reviewing and providing feedback to the Distribution. You also agree to cause this notice, along with the accompanying consent, to be included on all copies (or portions thereof) of the Distribution. The OSGi Alliance also grants you a perpetual, non-exclusive, worldwide, fully paid-up, royalty free, limited license (without the right to sublicense) under any applicable copyrights, to create and/or distribute an implementation of the Distribution that: (i) fully implements the Distribution including all its required interfaces and functionality; (ii) does not modify, subset, superset or otherwise extend the OSGi Name Space, or include any public or protected packages, classes, Java interfaces, fields or methods within the OSGi Name Space other than those required and authorized by the Distribution. An implementation that does not satisfy limitations (i)-(ii) is not considered an implementation of the Distribution, does not receive the benefits of this license, and must not be described as an implementation of the Distribution. "OSGi Name Space" shall mean the public class or interface declarations whose names begin with "org.osgi" or any recognized successors or replacements thereof. The OSGi Alliance expressly reserves all rights not granted pursuant to these limited copyright licenses including termination of the license at will at any time.

EXCEPT FOR THE LIMITED COPYRIGHT LICENSES GRANTED ABOVE, THE OSGi ALLIANCE DOES NOT GRANT, EITHER EXPRESSLY OR IMPLIEDLY, A LICENSE TO ANY INTELLECTUAL PROPERTY IT, OR ANY THIRD PARTIES, OWN OR CONTROL. Title to the copyright in the Distribution will at all times remain with the OSGi Alliance. The example companies, organizations, products, domain names, e-mail addresses, logos, people, places, and events depicted therein are fictitious. No association with any real company, organization, product, domain name, email address, logo, person, place, or event is intended or should be inferred.

THE DISTRIBUTION IS PROVIDED "AS IS," AND THE OSGi ALLIANCE (INCLUDING ANY THIRD PARTIES THAT HAVE CONTRIBUTED TO THE DISTRIBUTION) MAKES NO REPRESENTATIONS OR WARRANTIES, EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, NON-INFRINGEMENT, OR TITLE; THAT THE CONTENTS OF THE DISTRIBUTION ARE SUITABLE FOR ANY PURPOSE; NOR THAT THE IMPLEMENTATION OF SUCH CONTENTS WILL NOT INFRINGE ANY THIRD PARTY PATENTS, COPYRIGHTS, TRADEMARKS OR OTHER RIGHTS.
NEITHER THE OSGi ALLIANCE NOR ANY THIRD PARTY WILL BE LIABLE FOR ANY DIRECT, INDIRECT, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF OR RELATING TO ANY USE OR DISTRIBUTION OF THE DISTRIBUTION.

Implementation of certain elements of this Distribution may be subject to third party intellectual property rights, including without limitation, patent rights (such a third party may or may not be a member of the OSGi Alliance). The OSGi Alliance is not responsible and shall not be held responsible in any manner for identifying or failing to identify any or all such third party intellectual property rights.

The Distribution is a draft. As a result, the final product may change substantially by the time of final publication, and you are cautioned against relying on the content of this Distribution. You are encouraged to update any implementation of the Distribution if and when such Distribution becomes a final specification.

The OSGi Alliance is willing to receive input, suggestions and other feedback (“Feedback”) on the Distribution. By providing such Feedback to the OSGi Alliance, you grant to the OSGi Alliance and all its Members a non-exclusive, non-transferable,

worldwide, perpetual, irrevocable, royalty-free copyright license to copy, publish, license, modify, sublicense or otherwise distribute and exploit your Feedback for any purpose. Likewise, if incorporation of your Feedback would cause an implementation of the Distribution, including as it may be modified, amended, or published at any point in the future ("Future Specification"), to necessarily infringe a patent or patent application that you own or control, you hereby commit to grant to all implementers of such Distribution or Future Specification an irrevocable, worldwide, sublicenseable, royalty free license under such patent or patent application to make, have made, use, sell, offer for sale, import and export products or services that implement such Distribution or Future Specification. You warrant that (a) to the best of your knowledge you have the right to provide this Feedback, and if you are providing Feedback on behalf of a company, you have the rights to provide Feedback on behalf of your company; (b) the Feedback is not confidential to you and does not violate the copyright or trade secret interests of another; and (c) to the best of your knowledge, use of the Feedback would not cause an implementation of the Distribution or a Future Specification to necessarily infringe any third-party patent or patent application known to you. You also acknowledge that the OSGi Alliance is not required to incorporate your Feedback into any version of the Distribution or a Future Specification.

I HEREBY ACKNOWLEDGE AND AGREE TO THE TERMS AND CONDITIONS DELINEATED ABOVE.

0.2 Trademarks

OSGi™ is a trademark, registered trademark, or service mark of the OSGi Alliance in the US and other countries. Java is a trademark, registered trademark, or service mark of Oracle Corporation in the US and other countries. All other trademarks, registered trademarks, or service marks used in this document are the property of their respective owners and are hereby recognized.

0.3 Feedback

This document can be downloaded from the OSGi Alliance design repository at https://github.com/osgi/design The public can provide feedback about this document by opening a bug at https://www.osgi.org/bugzilla/.

0.4 Table of Contents

0.5 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.`

0.6 Revision History

The last named individual in this history is currently responsible for this document.

Revision	Date	Comments
0.1	April 2018	Initial version
0.2	April 2018	Update after feedback during Sofia F2F, with new use cases from Tim Ward, Tim Verbelen and Carsten Ziegeler. (David Bosschaert)
0.3	June 2018	Editorial changes (David Bosschaert).
0.4	July 2018	Input from Todor Boev, add roles, feedback from Washington F2F (David Bosschaert)

1 Introduction

OSGi has become a platform capable of running large applications for a variety of purposes, including rich client applications, server-side systems and cloud and container based architectures. As these applications are generally based on many bundles, describing each bundle individually in the application definition becomes unwieldy once the number of bundles reaches a certain level.

Furthermore, OSGi has no mechanism to describe other elements of the application definition, such as configuration or custom artifacts.

This RFP introduces the requirements for a higher level to describe OSGi applications that encapsulates the details of the various components that the application is built up from. It allows the description of an entire OSGi-based application based on reusable components and includes everything related to this application, including configuration, framework properties, capabilities, requirements and custom artifacts.

2 Application Domain

When developing large enterprise applications it is often the case that very few people know the role of every bundle or configuration item in the application. To keep the architecture understandable a grouping mechanism is needed that allows for the representation of parts of the application into larger entities that keep reasoning about the application manageable. In such a domain members of teams spread across the organization will need to be able to both develop new parts for the application as well as make tweaks or enhancements to their respective parts such as adding configuration and resources or changing one or more bundles relevant to their part of the application.

The higher level constructs that define the application should be reusable in different contents, for example if one team has developed a component to handle job processing, different applications should be able to use it, and if needed tune its configuration or other aspects so that it works in each setting without having to know each and every detail, bundle etc that the job processing component is built up from.

This RFP aims solving the problem of defining (large) applications in OSGi in a way that’s easy for humans and teams.

2.1 Relation to existing OSGi specifications

2.1.1 Subsystems Specification

While some might say that subsystems were designed for the purposes outlined in this RFP, subsystems are rather a possible way to implement the runtime realization of some aspects of the features. Subsystems are lacking authoring support and don't provide an architect-friendly design-time source format. Additionally, subsystems are limited to bundles, features often additionally declare configuration, custom content and custom metadata. Experience has shown that while subsystems work, authors of large systems find it difficult to work directly with these.

2.1.2 Deployment Admin Specification

The Deployment Admin specification also defines a deployable application format. These deployables are somewhat limited in that multiple deployment admin applications cannot have overlapping bundles, making this specification not very useful as many applications share certain dependencies. Additionally, the Deployment Admin specification does not define a format to architect features.

2.1.3 Application Admin Specification

The Application Admin Specification allows the deployment and management of Applications in OSGi. This specification is primarily aimed at UI-based applications. While this application provides a run-time API for deployment and management of applications, it does not provide a way to model features and applications for a systems architect.

2.2 Relation to existing Open Source solutions

A number of existing solutions exist both in Open Source as well as in closed source. From the Open Source space Apache Karaf Features are popular, as well as Eclipse Features. Additionally Apache Felix Bundle Archives provide a mechanism that could be used to deploy features.

Apache Sling Features provide a way to design and run features using JSON.

Bnd provides a mechanism to create an application runfile from a set of seed bundles, matching requirements against capabilities provided through one or more repositories.

Knowledge of the existing solutions is used to influence the requirements in this document.

2.3 Roles

The following section outlines roles involved in the creation of Feature-based OSGi applications. Note that different roles may be performed by the same individual.

Bundle Developer – A Bundle Developer writes OSGi bundle code. The Bundle Developer typically has a small scope and focuses on individual bundles or a small number of bundles that provide a cohesive piece of functionality.

Feature Developer – A Feature Developer creates OSGi features by collecting multiple bundles together to create higher level components.

Application Architect – The Application Architect designs a product by putting a number of high-level components together in a document. She defines the interaction between components in the product, and the external interactions of the product.

Application Assembler – The Application Assembler takes the input from the Application Architect and maps it to available features and configuration. He creates a high-level feature representing the application from existing features with added configuration.

Application Deployer/Administrator – The Application Deployer takes the feature created by the Application Assembler and turns it into a runnable application. He does this by mapping all requirements to capabilities and by resolving all version ranges to a specific version range. He sets configuration to integrate with external systems such as databases, external microservices and others. He then runs the application on his infrastructure.

Quality Engineer – A Quality Engineer needs to test an application before it's released. The QE may also be asked to ensure that the application still works when one or more individual bundles or features, configuration or other resources are replaced with different ones.

2.4 Terminology + Abbreviations

Application – An application is a complete definition of an entire runtime, including the specific versions of all of its dependencies and transitive dependencies. It also includes custom artifacts, such as data files and configuration. An application may have a number of configuration elements specified at startup.

Feature – A feature combines a number of bundles together to provide a logical piece of functionality. Features may also include other features, configuration and other artifacts. Different than for applications, features may use version ranges for dependencies.

Different forms of features exist:

Reusable Feature – A feature which contains a number of root bundles and possibly some root requirements, but as little as possible or no binding of these root requirements to providers. This allows the feature to be used in a variety of contexts.

Runnable Feature – A feature which is fully resolved, having all its requirements satisfied and bound to capabilities.

3 Problem Description

OSGi has no support for describing large applications. Application developers need to come up with their own way to do this. When applications are getting larger and are developed by multiple teams this becomes a challenge, especially in cases where the application is composed of multiple features each of which are groups of bundles, configuration, metadata and other artifacts.

4 Use Cases

This section contains a number of use-cases. Although some use-cases appear to be similar, they are not identical, to ensure completeness they are kept here.

4.1 Author a Feature

Linda is designing high-level components for a system she is building. She does this by selecting existing bundles, features, custom resources and configuration together and building this into features. The features also provide capabilities that are not provided by any of the individual components themselves. Linda would like an easy-to-use mechanism to author the feature definitions and when it's done she's planning to upload the features to a Maven Repository.

4.2 Higher Level Building Blocks

Joe would like to create a higher level building block from lower level ones. For example, he would like to specify a building block providing a messaging system, which might have an API bundle, an implementation bundle, some library bundles required by the implementation and OSGi configuration to setup the building block (at least an initial configuration). Finally he would like to publish the result to a repository so that it can be re-used in other building blocks.

4.3 Generate an application from a number of features

Ross needs to assemble an application from a number of features developed across his company and in open source. He would like to refer to the features that together compose the application through Maven coordinates and he would like to get an application definition where each bundle that is ultimately used is exactly defined in order to provide total predictability over what the application is composed at run time. Ross needs to be able to take the application definition and pass this to a launcher tool that runs it for him.

4.4 Create a Docker image for a specific feature

Janet is composing a multi-node application as a number of Docker containers. Each node provides part of her application and can be independently scaled. Nodes are defined as OSGi feature definitions so she needs to be able to take a feature definition and turn this into a Docker image.

4.5 Provide 'most' of the configuration, some at launch

Barry is using an application that has been created from a number of features. The features provided the default configuration needed to use the application, but Barry needs to supply additional configuration information at launch time, such as database credentials.

4.6 Add custom metadata

Sarah is a Feature developer who wishes to record additional performance testing characteristics in the feature model. With this information a testing process can check that the performance of the feature is adequate and if not take appropriate measures. She wants to specify this performance information per external API provided through the Feature.

4.7 Test an application with a patched bundle or configuration

The testing team is tasked with verifying that updating a specific bundle in an existing application does not cause any regressions. For this, the testing team needs to update the application, which is defined through a number of features with one specific bundle. The testing team needs to be able to do this without having to modify the original features or application, since these have already been released and cannot be modified any more.

4.8 Remote Services

Susan wants to assemble an application from a set of features. The application contains a bundle which requires a FooService which performs complex calculations. This service is available as an OSGi remote service on a High Performance Computing solution, and so rather than deploying a FooService locally Susan would like to assemble her application using the remote service. The features that she chooses must assemble successfully even though no local bundle provides the FooService capability.

4.9 Resolve a Distributed Application

The devops team wants to define which features will be deployed on what nodes in a distributed system. They want to use the resolver to calculate, for each node that uses a feature, whether it only needs the API using Remote Services to access the remote service or whether it also needs the implementation. The team is using the OSGi ClusterInfo specification to expose what is available on the various nodes in the cluster, in terms of OSGi Capabilities and Requirements.

4.10 Supply a Plugin to an existing Application

Harry is a developer for Bnd. Harry wants to add Microprofile-Config support to Bnd. To do this Harry must implement a Bnd plugin that understands and reacts to the @ConfigProperty annotation. Harry implements the support as a bundle with some additional resources. He provides the complete solution as a plugin to Bnd by adding an OSGi feature to the Bnd runtime.

Harry is a plugin developer for an OSGi-based application. The application is extensible and can be customized by providing additional features, which contain a number of bundles, configuration and other artifacts. The feature provides a clean mechanism to describe all the resources required by the application's plugin. Harry would like to share his plugin with other users of the application as it would benefit them too.

5 Requirements

The feature model is about describing a feature, aggregating features to either build higher level features or an application. The model should meet the following requirements:

- FM010 - The feature model should be described through a text format which is easily consumable by both humans and machines, that can be edited with common editors and support text-based diff operations.
- FM020 - A feature must be describable through a single file.

- FM030 - Multiple features must be described in multiple files.
- FM040 - The feature model language must support comments.
- FM050 - The feature model may support more than one text-based definition language where the language used can be easily inferred, for example from the file extension.
- FM060 - The feature model should provide support for long and multi-line values without creating files that become hard to handle.
- FM070 - A feature model must have a unique identifier.
- FM080 - A feature model must have a version.
- FM090 - A feature model must be referenceable through Apache Maven coordinates.
- FM100 - It must be possible to specify the bundles belonging to the feature, including version.
- FM110 - It must be possible to specify the bundles in a feature in terms of Apache Maven coordinates.
- FM120 - The feature model must allow the specification of the order in which the bundles inside the feature are started. This should be relative to when the feature itself is started.
- FM130 - It must be possible to define whether a bundle is mandatory or optional.
- FM140 - It must be possible to associate any additional metadata like a hash with a bundle.
- FM150 - It must be possible to specify the OSGi configurations for a feature.
- FM160 - Both normal OSGi configurations as well as factory configurations must be supported. The feature model must support all data types supported by the OSGi Configuration Admin specification.
- FM170 - The OSGi configuration resource format as defined in the OSGi Configurator Specification must be supported.
- FM180 - It must be possible to associate an OSGi configuration with a bundle within a feature. If the bundle is not resolved at runtime then the associated configuration also does not get installed.
- FM190 - It must be possible to define framework properties.
- FM200 - The feature model must be extensible to allow other artifacts than bundles.
- FM210 - It must be possible to specify the artifacts in a feature in terms of Apache Maven coordinates.
- FM220 - It must be possible to associate any additional metadata like a hash with an artifact.
- FM230 - It must be possible to define whether an artifact is mandatory or optional.
- FM240 - The feature model must be extensible to allow other/additional content.
- FM250 - It must be possible to mark the additional content as optional.

- FM260 - A feature must be able to specify additional requirements and capabilities that extend the requirements and capabilities from the contained artifacts.

- FM270 - A feature must be able to extend other features.

- FM280 - A feature must be able to depend on other features through the requirements/capabilities model based on the feature contents. The feature model must be able to deal with circular dependencies. However, there must be no way of explicitly requiring a feature from another feature.

- FM290 - The feature model must describe how several features are aggregated to build a higher level feature. This description must include all parts of the feature model (bundles, configurations, framework properties etc.). The description should be general for extension, which means it should describe how extensions are aggregated without requiring the model implementation to know the type of extension.

- FM300 - The feature model must describe how several features are combined to build an application. This description must include all parts of the feature model (bundles, configurations, framework properties etc.). The description should be general for extension, which means it should describe how extensions are aggregated without requiring the model implementation to know the type of extension.

- FM310 - When features are aggregated, either to create a higher level feature or an application, and a bundle/artifact is encountered with different versions, the feature model must be capable of only using the bundle/artifact with the highest version number. The detection is based on the artifact/bundle id, not the bundle symbolic name.

- FM320 - When features are aggregated, either to create a higher level feature or an application, and a bundle/artifact is encountered with different versions, the feature model must be capable of including both versions side-by-side. The detection is based on the artifact/bundle id, not the bundle symbolic name.

- FM330 - When features are aggregated, either to create a higher level feature or an application, the resulting feature or application must be minimal meaning it must not contain additional or unneeded artifacts.

- FM340 - The feature model must calculate the startup order of bundles for an aggregated application respecting the dependencies between features and their contents.

- FM350 - The feature model must support variables to be used throughout the model, avoiding the need to repeat the same value several times.

- FM360 - When features are aggregated, the ordering of the processing of those features needs to be predictable and stable.

- FM370 - The feature model must support adding or overwriting requirements and capabilities of a contained bundle or artifact. This is in oder to correct invalid metadata or to add missing metadata of the artifact.

- FM380 - The feature model must support adding or overwriting manifest headers for a bundle. For example to allow to change the bundle symbolic name or to add missing OSGi metadata to a plain jar file.

- FM390 - The feature model must support a textual representation for an application aggregated out of features. The format should be as similar as possible to the feature format.

- FM400 - It must be possible to specify the framework to launch an application as part of the application model.

- FM410 - When features are aggregated to either a higher level feature or an application, the resulting feature or application must still contain the variables.
- FM420 - The startup order of features and bundles must be part of the resulting aggregated application model.
- FM430 - The feature model must support additional, optional information about the feature like a human readable title, a description, vendor and licensing information.
- FM440 - The feature model must use a versioned descriptor format so that if the format evolves in the future users can state in feature model files what version they are written for.
- FM450 - The feature model should provide an externally accessible API for reading and writing feature files.
- FM460 - It must be possible to add new features to an existing application by placing additional files in a container's file system.
- FM470 - It must be possible to alter existing features in an application by placing additional files in the file system. For example to uninstall or update a bundle provided by an existing feature or to alter a configuration set by an existing feature.

6 Document Support

6.1 References

[1]. Bradner, S., Key words for use in RFCs to Indicate Requirement Levels, RFC2119, March 1997.

[2]. Software Requirements & Specifications. Michael Jackson. ISBN 0-201-87712-0

6.2 Author's Address

Name	David Bosschaert
Company	Adobe
Address	
Voice	
e-mail	bosschae@adobe.com

Name	Todor Boev
Company	Software AG
Address	
Voice	
e-mail	

Name	Carsten Ziegeler
Company	Adobe
Address	
Voice	
e-mail	

Name	Tim Verbelen
Company	Ghent University
Address	
Voice	
e-mail	

Name	Tim Ward
Company	Paremus
Address	
Voice	
e-mail	

Name	
Company	
Address	
Voice	
e-mail	

Name	
Company	
Address	
Voice	
e-mail	

6.3 End of Document