RFC 242 - Condition Service

Draft

12 Pages

Abstract

Declarative Services provide a nice and powerful method to give activation conditions for a Service or Component. The Condition Service attempts to extend this, in order to support more complex scenarios, that can't be handled by the current mechanism.

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 10.1.

`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
Initial	*2019/04/11*	*Initial Version* *Jürgen Albert, Data In Motion Consulting GmbH, j.albert@data-in-motion.biz*
Version 0.1	*2019/05/22*	*Moved using Conditions as enablement criteria to considered alternatives. Every Component now needs an implicit Conditon.* *Clarified Configurable Condition Factory.* *Jürgen Albert, Data In Motion Consulting GmbH, j.albert@data-in-motion.biz*

1 Introduction

Declarative Services are a comfortable and powerful tool with which managing the life cycle of your services is easy and comfortable. The ability to define dependencies to other services together with target filters and cardinalities, make it unmatched by most other comparable mechanisms. Unfortunately the possibilities to set

conditions, under which DS should activate or deactivate a Component are limited. Thus this RFP will explore an extension to DS allowing for more complex scenarios.

2 Application Domain

Components have a clearly defined Lifecycle. Unfortunately the conditions when a component should be activated and eventually registered as a service, can be more complex then the general availability of the directly used services. Thus a Mechanism would be useful to allow a more precise and complex definition of conditions surrounding the Lifecycle of a Component

2.1 Terminology + Abbreviations

2.1.1 Conditions

DS purely depends on References and there provided conditions to determine if a component can be activated. Conditions should set of rules that gives DS, CDI or Services a clue, when a component can be enabled or disabled that are independent of the references a service requires.

3 Problem Description

Services and Components are a great way, to represent processes in the real world or more artificial things like business processes.

DS already allows the developers to define a Cardinality to each service reference, that gives clues, when a service should be activated and deactivated. The Configuration Admin extends this capability, so target filters can be assigned at configuration time. Additionally the amount of services to inject, can be specified more precisely then in the component xml or with the Annotations.

This RFC will explore the Use cases that exceed the possibilities DS gives the developer. The mechanism is not necessarily in the direct domain of DS, but can be picked up by it.

All the Use cases that will follow, can be achieved somehow by using e.g. DS to activate a Component, that than uses Servicetrackers, to check if the special needs are fulfilled. This is often uncomfortable, and the principle is usually the same.

4 Requirements

4.1.1 General

- G010 – It MUST provide a marker service that can be used as a mandatory injecte for any component.
- G020 – Everybody must be able to register its own Condition.
- G030 – There MUST always be a registered True Condition.
- G040 – Every registered Condition MUST have a persistent identifying property, that can be targeted by other components.
- G060 - The Condition must be supported in all injection models.
- G070 – The state of a Condition and its changes MUST be observable.

4.1.2 Configuration of Conditions

- C010 – The Solution MUST provide a mechanism to create and program conditions via configuration.
- C020 – It MUST define a domain specific language to program Conditions. It must support boolean logic and cardinality.
- C030 – The configurable component MUST provide a DTO showing the state of its requirements.

4.1.3 Declarative Services

- DS010 – DS MUST support a mechanism to bind the enablement of a component to a Condition.
- DS020 - Every DS Component by default must depend on the True Condition of G030.
- *DS030 – CDI MUST meet the Requirements DS010 and DS020 as well.*

5 Technical Solution

This solution proposes extending the core to support Conditions and even entails changes to DS and CDI, that need to be honored by any other future Component Model, except Blueprint.

5.1 Condition

A valid Condition is a Service registration with the `org.osgi.service.condition.Condition` Interface as an exposed ObjectClass, together with a unique Condition id property (`org.osgi.service.condition.Condition#CONDITION_ID`). A Condition does not offer any functionality and acts just as a clearly identifiable Marker.

The proposed Interface is:

```
@ConsumerType
public interface Condition {

	public static final String CONDITION_ID = "osgi.condition.id";

	public static final String DEFAULT_TRUE_CONDITION_ID = "true";

	/**
	* A default instance that can be used to register Conditions,
	* without cluttering the JVM with useless Object instances.
	*/
	public static final Condition INSTANCE = new ConditionImpl();

	/**
	 * A default Implementation for the {@link Condition} interface
	 */
	static class ConditionImpl implements Condition{

	}


}
```

A Component that wishes to announce a Condition has now 3 possibilities:

1. A Component implements the `org.osgi.service.condition.Condition` Interface and announce it as an Servicetype of its own Service Registration, together with a `Condition.CONDITION_ID` property.

2. Create an own ServiceRegistration, using for example the `Condition.INSTANCE`, together with a `Condition.CONDITION_ID` property.

3. Create an own ServiceRegistration, using any Object implementing the `Condition` Interface, together with a `Condition.CONDITION_ID` property.

5.1.1 The default TRUE Conditions

The Framework itself will always register a TRUE Condition instance, that always can be relied upon. This Condition must have the `Constants.CONDITION_ID property` with the value “true”.

5.2 Condition Factory

In order to create more complex Conditions easily a configurable Factory is called for. This allows integration of Conditions in a Systems without the requirement to actually register a Condition Service.

The implementation implementation of the ConditionFactory~~A Bundle~~ should register a `org.osgi.service.cm.ManagedServiceFactory` with the service ~~name~~PID "osgi.condition.factory". A configuration it might receive via ConfigurationAdmin can contain two lists of target filter. One for filters that MUST find at least one Service each in order to have a Condition registered. The other one represents a List of filters, that MUST NOT find any matching service. The moment all the included and non of the exclude filters match a Service, a Condition is registered with the given identifier and the additional properties. If any of the include Services go away or an excluded Service becomes available, this Condition must be unregistered.

List of Properties:

Name	Value
`osgi.condition.identifier`	The `Constants.CONDITION_ID` that will be set to the condition that will be registered ~~when~~if all filters are satisfied
`osgi.condition.properties.*`	Properties like this will be registered with the condition ~~when~~if all filters are satisfied. The key will be the * part.
`osgi.condition.~~includes~~match.all`	An optional list of valid target filters. A Condition will be registered ~~when~~if each filter finds at least one matching service.
`osgi.condition.``match.none`~~**`excludes`**~~	An optional list of valid target filters. A Condition will not be ~~un~~registered ~~when~~if any filter finds at least one matching service.

An Example Configuration utilizing the Configurator would look like as follows:

```
{
    ":configurator:resource-version": 1,

    "osgi.condition.factory~test" : {
       "osgi.condition.identifyer" : "resulting.condition.id",
       "osgi.condition.properties.custom.condition.prop" : "my.property",
       "osgi.condition.~~includes~~match.all" : [
              "(&(objectClass=org.foo.Bar)(my.prop=foo))",
              "(my.prop=bar)"
       ],
       "osgi.condition.~~excludes~~match.none" : [
              "(&(objectClass=org.foo.Fizz)(my.prop=buzz))"
       ]
    }
}
```

5.3 Support in DS and CDI

Conditions must be an implicit service reference, that will always be there. It can be made explicit and modified with some limitations described later.

To present all its mandatory attributes of the implicit Reference, the following example will show it as a DS Component Annotation:

```
@Component(name = "A Exampelle Component", reference = {
	@Reference(
	name="osgi.ds.activation.condition",
	target = "(osgi.conditon.id=true)",
	cardinality = ReferenceCardinality.MANDATORY,
	policy = ReferencePolicy.DYNAMIC,
	service = Condition.class,
	policyOption = ReferencePolicyOption.RELUCTANT)})
```

By default, it thus will wait for the default true Condition.

The activation conditions target Filter can be changed by setting the component property "org.osgi.ds.activation.condition.target" to a valid Filter expression. This will delay the activation of the component until a Condition matching the given filter becomes available.

The implicit Reference can become an explicit reference as shown with the following Example:

```
@Component
public class MyComponent {

	@Reference(name="osgi.ds.activation.condition")
	private Condition defaultCondition;

}
```

TODO: Make the example a bit more expressive.

It can be used with any of the known injection points, similar to any other Service. It is not permitted to use the Reference name “org.osgi.ds.activation.condition” with any other type then Condition. Any other attribute can be changed at the users discretion and at his own risk.

To allow the convenient change of the default activation target, the following Annotation will be provided:

```
@ComponentPropertyType
@Retention(RetentionPolicy.CLASS)
@Target(ElementType.TYPE)
@RequireServiceComponentRuntime
public @interface DefaultActivationCondition {
	/**
	 * Prefix for the property name. This value is prepended to each property
	 * name.
	 */
	String PREFIX_ = "osgi.ds.activation.condition";

	/**
```

```
	 * Service property providing a new target filter for the default activation
	 * Condition
	 *
	 * @return The target filter for the default activation condition.
	 */
	String target();
}
```

6 Data Transfer Objects

The default condition dependency reference need to be visible in the respective DTOs.

DTOs are not necessary. A Condition is just a Service. All required Information are already available through the DTOs of DS and CDI,

7 Javadoc

Please include Javadoc of any new APIs here, once the design has matured. Instructions on how to export Javadoc for inclusion in the RFC can be found here: https://www.osgi.org/members/RFC/Javadoc

8 Considered Alternatives

8.1 Use conditions to trigger Components enablement

This notion was abandoned right from the start, because a Condition is just a Service and DS and CDI have already mechanisms in place to react and require to them.

Currently Components have an enabled flag, to tell DS to handle a Component. A component can disable itself via its Components Context. The only possibility to enable a Component again would be to hand the `ComponentContext` to someone else, so he can enable it again or via the ServiceComponentRuntime Service.

Conditions can be a more powerful and convenient way for a component developer, to control its enablement. Thus a new enablement condition filter will be used beside the enable marker.

8.1.1 XML Schema

To use the condition as trigger, a new field `enablementCondition` is added to the component XML schema of type String. The filed is optional and must be a valid filter if set.

Do we want to mark the enable field as deprecated or do we want to keep it forever?

8.1.2 Annnotation

The @Component Annotation must be amended with the following field:

```
String enablementCondition() default "";
```

This will be mapped to its counterpart in the component XML at build time.

8.1.3 Behavior

DS must adhere to the following rules:

	State	Behavior
1	`enable=true enablementCondition=<empty>`	DS enables the Component, the moment the default true Condition becomes available (see 5.1.1).
2	`enable=true enablementCondition=”(foo=bar)”`	DS enables the Component, the moment a condition matching the given filter becomes available.
3	`enable=false enablementCondition=<empty>`	The Component is disables. If the enable state is changed via DTOs or ComponentContext the behaviour of 1 applies.
4	`enable=false enablementCondition=”(foo=bar)”`	The Component is disables. If the enable state is changed via DTOs or ComponentContext the behaviour of 2 applies.

9 Security Considerations

Description of all known vulnerabilities this may either introduce or address as well as scenarios of how the weaknesses could be circumvented.

10 Document Support

10.1 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

Add references simply by adding new items. You can then cross-refer to them by chosing <Insert><Cross Reference><Numbered Item> and then selecting the paragraph. ***STATIC REFERENCES (I.E. BODGED) ARE NOT ACCEPTABLE, SOMEONE WILL HAVE TO UPDATE THEM LATER, SO DO IT PROPERLY NOW.***

10.2 Author's Address

Name	Jürgen Albert
Company	Data In Motion Consulting GmbH
Address	Kahlaische Str.4, 07745 Jena, Germany
Voice	+49 1577 2521634
e-mail	j.albert@data-in-motion.biz

10.3 Acronyms and Abbreviations

10.4 End of Document