



OSGiTM
Alliance

RFP-193-Condition-Service

Final

6 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.

Copyright © OSGi Alliance 2019.

All company, brand and product names contained within this document may be trademarks that are the sole property of the respective owners.
The above notice must be included on all copies of this document that are made.

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.....	3
2 Application Domain.....	3
2.1 Terminology + Abbreviations.....	3
2.1.1 Activation Conditions.....	4
3 Problem Description.....	4
4 Use Cases.....	4
4.1 Service unavailable.....	4
4.2 Conditions are only known at configuration but not at development time.....	5
4.3 Service injections after activate was called.....	5
5 Requirements.....	5
5.1.1 General.....	5
6 Document Support.....	5
6.1 References.....	5
6.2 Author's Address.....	6
6.3 End of Document.....	6

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.

Revision	Date	Comments
Initial	25.08.2018	<i>Initial Jürgen Albert</i>
V01	28.01.2019	<i>Renamed to Condition Services Rework of the Requirements</i>

1 Introduction

Declarative Services are a comfortable and powerful tool with which managing the lifecycle 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 RFP 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 Use Cases

4.1 Service unavailable

Anton Develops a system, that is a on premise solution that a customer installs via Docker. Every installation specific configuration will be handled via a webfrontend after the first start. The system supports a couple of different login mechanisms, that utilize servlet filters. As long as no login filter is registered a default login filter must be available and only if no other login service is registered.

4.2 Conditions are only known at configuration but not at development time

Betty develops a Service that handles billing in a Webshop. The service consumes other services with the IBillingProvider Interface, that is available in a couple of implementations for e.g. Debit, VISA and PayPal. The Webshop is home to different tenants, which get the allowed billing services according to his individual payment plan. Thus Bettys Services will be configured at runtime according to the tenants account and must be activated only when e.g. the VISA and Debit is available.

4.3 Common indirect preconditions

Carl develops an RCP Application, where a lot of its components indirectly depend on the availability of the ExtensionRegistry and SWTApplication. He wants that his components only get enabled and activated, when this services are up and running.

4.4 ConfigurationPlugins for ConfigurationAdmin

Dorothy has a Configuration Plugin for the ConfigurationAdmin, that sets certain passwords from a secure storage to internal configurations. In order to make her System work, she must make sure, that the ConfigurationAdmin waits to apply her configurations until her Plugin is registered.

5 Requirements

5.1.1 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 service.
- G030 – There MUST always be a registered service acting a marker for a True condition.
- G040 – Every registered Condition service must have a unique identifying property, that can be targeted by other components.
- G060 - The Condition must be supported in all injection models.

5.1.2 Configuration of Conditions

- C010 – The Solution MUST provide a mechanism to create conditions via configuraton.
- C020 – It MUST be possible to define an undefined amount of filters via configuration, that delays activation of a component until the conditions are met.
- C030 – It MUST be possible to define an undefined amount of filters via configuration, that cause deactivation of a component when the conditions are met.

5.1.3 Declarative Services

- DS010 – DS MUST support a mechanism to define a filter for a Condition to trigger a components activation or deactivation.
- DS020 - Every DS Component by default must depend on the True Condition of G030.
- DS030 - The default Condition dependency must be configurable via Config Admin (I added this in our F2F in Jena and I'm not 100% sure what we had in mind here. It reads a bit like: change the configuration of the default true condition from DS020 to any other condition)

5.1.4 Configurator/ConfigurationAdmin

- C010 – The Configurator/ConfigurationAdmin MUST be able to handle and honor a filter for a condition.

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	Jürgen Albert
Company	Data In Motion Consulting GmbH
Address	Kahlaische Str. 4, 07743 Jena, Germany
Voice	+49 1577 2521634
e-mail	j.albert@data-in-motion.biz

6.3 End of Document