RFP 156 – Native OSGi

Draft

14 Pages

Abstract

Native OSGi is an effort to revive parts of RFP 89 – Universal OSGi. This RFP specifically focusses on the possibility of implementing an OSGi framework in C and C++. The goal is, to be as close to the OSGi Core specification as possible, without restraining the natural usage of the language itself. Interoperability between C and C++ is important and will receive special attention.

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
Initial	*10/24/12*	*Initial document* *Alexander Broekhuis, Luminis, alexander.broekhuis@luminis.eu*
Update	*03/12/13*	*Updates after reviewing the first draft during a OSGi meeting.* • *Added additional authors (Sascha and Steffen)* • *Added overview of current eco-system and frameworks* • *Detail library usage and platforms* • *Verify/rewrite requirements* *Alexander Broekhuis, Luminis, alexander.broekhuis@luminis.eu*
Update	*03/16/13*	*nOSGi specific updates.* • *nOSGi specific updates* • *Added related work* • *Module layer updates* • *Use Cases* *Steffen Kächele, University of Ulm, steffen.kaechele@uni-ulm.de*

Revision	Date	Comments
Update	*03/22/13*	*Updates:* • *CTK Plugin Framework specific updates* • *Added CppMicroServices* • *Module layer updates* *Sascha Zelzer, German Cancer Research Center, s.zelzer@dkfz-heidelberg.de*
Update	*10.04.13*	*Comments from Cologne F2F* *Alexander Broekhuis, Luminis, alexander.broekhuis@luminis.eu*
Update	*11.04.13*	*Comments from Cologne F2F* *Steffen Kächele, University of Ulm, steffen.kaechele@uni-ulm.de*

1 Introduction

The OSGi specification currently exclusively focuses on the Java Runtime and languages. In the past RFP 89 – Universal OSGi has been started to extend this scope to different languages; C++, C# and ActionScript/JavaScript. Even though RFP 89 is still relevant, is has a very broad scope. This RFP is started to limit the scope to the native languages (C and C++). From this point of view, RFP 89 can be seen as an umbrella RFP.

Many of the reasons mentioned in RFP 89 still apply, amongst others:

- Broaden scope of OSGi in other domains
- Ease the introduction of OSGi concepts in non-Java environments

Since the initiation of RFP-89 a lot has changed, and several Native OSGi-like implementations have been written. This RFP is an effort of several of these projects with a common goal:

- Combine the effort to reach a greater audience
- Write a common specification which makes it possible to share bundles

Even though the scope of the RFP is limited to C and C++, there are many other languages and compilers that create binaries for the native runtime. While the focus will be on C and C++, the requirements should try to be

open enough for these other languages. In other words, requirements should not force solutions which would block other languages that use the same runtime.

1.1 Participants

At this moment the following projects are participating:

- Apache Celix – A C implementation based on OSGi 4.3 with a focus on Java interoperability and remote services.
- CTK Plugin Framework – A C++ implementation based on OSGi 4.3 using the Qt toolkit. Focuses on adhering to the Core API as close as possible.
- NOSGi – A C++ implementation based on OSGi 4.* with a unique bundle wiring mechanism and a focus on embedded devices.
- CppMicroServices – A C++ implementation based on the ideas of “OSGi Lite”, i.e. without the dynamics of the module layer. Its focus is on a type-safe service layer API in C++.

1.2 Existing Eco-System

These participating projects already provide a fairly extensive collection of services and bundles. The expectation is that these bundles will be adapted to be compatible with the Native-OSGi specification. This provides an extensive eco-system that can and will be leveraged by the Use-Cases described in chapter 4.

The following overview lists all the bundles/services provided by the different participating projects:

- Apache Celix
  - OSGi Compendium Chapter 101 - Log Service
  - OSGi Compendium Chapter 103 - Device Access
  - OSGi Enterprise Chapter 122 – Remote Service Admin
  - Log Writer
  - Shell Service
  - Shell Textual User Interface (TUI)
  - Dependency Manager (based on the Apache Felix Dependency Manager)
- CTK Plugin Framework
  - OSGi Compendium Chapter 101 – Log Service
  - OSGi Compendium Chapter 104 – Configuration Admin
  - OSGi Compendium Chapter 105 – MetaType Service
  - OSGi Compendium Chapter 113 – Event Admin

- nOSGi
  - Support for Module Layer
  - Shell Service
  - Remote Shell Service
  - Job Scheduling Service (i.e. a cron bundle)
  - Bus Access Service (i.e. an 1-wire bundle)

1.3 Technical Background

The participating projects already provide a working framework. Each implementation uses different solutions for the different problems. Within Native-OSGi several solutions will be combined to provide one common specification.

- Apache Celix
  - Bundle Format
  - C API
- nOSGi
  - Module Layer and Wiring model (versioning etc).
  - Some C++ interfaces
- CTK Plugin Framework
  - C++ API
  - LDAP Filter mechanisms
- CppMicroServices
  - Type-safe C++ service layer API

1.4 Module Layer

The Module Layer must provide features proposed by the OSGi core specification. The nOSGi project already presents solutions for many native specific issues. In Native-OSGi the Module Layer is composed out of libraries (e.g. ELF format on Linux, Mach-O format on MacOSX, and PE format on Windows). These libraries are located in bundles, and the module layer takes care of loading and wiring the different libraries.

Native OSGi should use shared libraries on the level of Java Packages.

- Have multiple libraries per bundle
- Have exported and private libraries

Just like Java Packages this makes it possible to make a distinction between the private implementation and the exported services.

- Allow code sharing

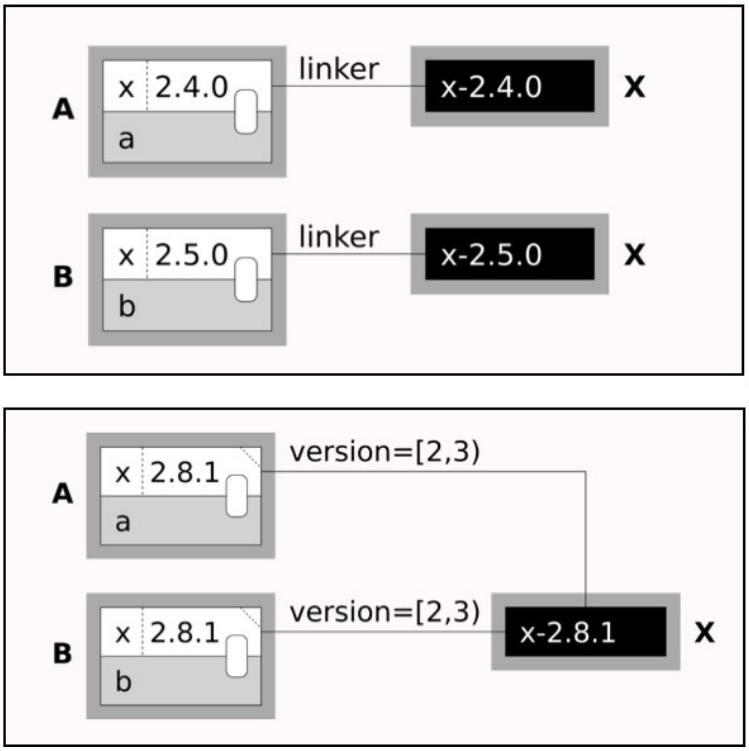
  Using exported libraries it is possible to share code between bundles.

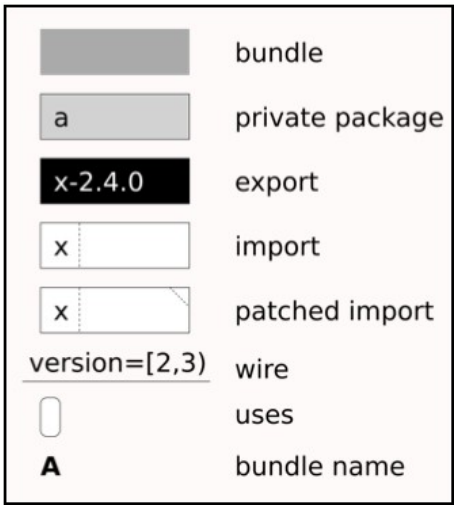
An important aspect of OSGi is versioning of packages, and the wiring needed to find the correct version of a dependency. Whereas Java by itself does not provide any default versioning mechanism, the different native linkers do. As proposed by nOSGi, in Native OSGi, libraries should be altered to match dependencies of OSGi wiring:

- Library dependencies are patched at runtime
- Allows multiple versions
- Allows bundle updates

The following diagram depicts the wiring process. In the upper diagram the situation at link-time is detailed. The library dependencies are hard-coded by the linker during the linking process. This results in a situation where Library A depends on version 2.4.0 of Library X, while Library B, developed with a different set-up, depends on version 2.5.0 of Library X.

If in the Meta-Data of the bundle providing library A or B the version range [2, 3) is provided, the module layer (resolver) looks for any bundles which fit in this range. Since the linker already has embedded the version info in the library itself, this information has to be updated. So the version of the providing bundle/library (found by the resolver) is used to patch the version required by library A and B.





In a Native OSGi setup, interoperability may be reduced when using compilers of different vendors. As a consequence, only bundles that were compiled with a compiler of the same vendor may share code. In most cases, bundles of the system are compiled with one certain compiler. Using an intermediate format such as LLVM or source bundles may allow to mix bundles that were compiled with different compilers (see [4].).

Alternative

Even though the model depicted above works, it might be worthwhile to investigate different solutions. A possible alternative is to extend the used linker (or linkers) to solve this. This would mean that instead of linking to a fixed version, the version range can be used. In this case there is no need to update the libraries at runtime. However, this might introduce restrictions to the target environment, e.g. a proper linker has to be supplied for every operating system, runtime environment and possible versions.

A much more complex solution might be the usage of one dedicated linker on all platforms. This linker can be bundled together with the framework which means the platform linker is ignored. Benefits of this are:

- No different library format, which means no different solutions (bundles or anything else) for different platforms.
- Only need for one common solution for wiring.

2 Application Domain

Native OSGi can be applied in any domain which has/wants to rely on native code. Common examples are:

- Embedded software vendors
- (Medical) Imaging solutions
- Sensor Networks

Since it focuses on the implementation of the Core Specification in native languages it is difficult to list the final application domain. A few usage examples are:

A software vendor has a large distributed (embedded) software stack, and wants to be able to dynamically (re)configure a running system. For this they need a module based system which has the ability to replace a module at runtime.

A software vendor has a mixed software base of Java and C/C++ software. The Java software already uses OSGi. To be able to leverage the same benefits, they want to have a similar solution for their native code.

Alternatives to Native OSGi are:

- CORBA and the CORBA Component Model

- Service Component Architecture (SCA)
- Open API Platform (OpenPlug) from Alcatel-Lucent

CORBA and the CORBA component model also proposes an alternative to OSGi. It was designed to support composition and deployment of applications on distributed resources. The component standard defines the interfaces (ports), dependencies, and rules managing interaction between components as well as their lifecycle. Yet, the tight coupling between code modules with respect to dependencies hinders flexibility and adaptability of applications.

Lifecycle management of Open-Plug is quite similar to the one proposed in Native OSGi. However, to run code on devices, it uses proprietary technology that interprets code. A Native OSGi implementation should use open standards and execute code natively on devices. Moreover, Native OSGi uses standard method invocation for inter-component communication, whereas Open-Plug uses a more heavyweight message bus.

2.1 Terminology + Abbreviations

N-OSGi - Native OSGi

3 Problem Description

3.1 Introduction

This Document intends to describe the requirements that are necessary to bring OSGi also into the native (C/C++) space. This problem is specifically mentioned in RFP-89 as “Native OSGi”. The following description is an update of the content from RFP-89.

“Having a Native OSGi will make it possible to use legacy C/C++ code and more easily than with JNI have it coexist with Java bundles. It would also make it possible to port Java bundles to more efficient C/C++ bundles. It is clear however, that a native OSGi will not support the full set of Java and/or OSGi R5 features (e.g. reflection).”

Even though this section mentions coexistence with Java bundles, this RFP will focus solely on the Core Specification. Interoperability with Java bundles can be achieved using eg Remote Services.

3.2 Rationale

The need for component based development is not new to the native world, several concepts have been introduced to solve these problems (CORBA, COM, etc). But very few have been successful. Over the years OSGi has proven itself to be a stable and accepted solution to this problem, but it focuses specifically on the Java runtime. In large projects, but also in the traditional application domain, native languages are still used extensively, also besides Java. Being able to use one common component framework for these domain would be a great win to such projects. For these projects it makes sense to be able to use OSGi concepts in a native environment.

For certain areas a binding to C/C++ is enough, but for projects with a greater focus on native languages it makes more sense to have an actual native implementation. For example, if 90% of the codebase is native, it doesn't make any sense to use a Java OSGi framework.

4 Use Cases

4.1 Dynamic (Re)Configuration [3].

A software vendor has a large distributed (embedded) software stack used for a sensor network. To be able to react to a changing environment it must be possible to reconfigure software. Since this can happen during actual usage this must be done dynamically and with little impact on the running network.

To be able to do this, a module based architecture is defined. This architecture will be realized using Native-OSGi. OSGi provides a clear component based model, as well as all aspects needed to be able to reconfigure the system at runtime (through the module and life cycle layer).

4.2 Updating and bug-fixing in home automation systems

A number of software modules for a security system or an home gateway router have been developed. The system should provide support for dynamical updates e.g. in term of bug-fixing or to add new functionality. End-users should further be able to extend their system with additional software (i.e. plugins) with no need to restart the entire platform. To keep the hardware costs low, software is typically developed in native languages and for exotic hardware architectures such as Mipsel. A native OSGi implementation can bring the dynamical aspects to these devices without increasing the Bill of Material.

4.3 High performance computing software

A developer wants to extend mathematical software with a new algorithm. He is able to implement his algorithm according to an existing interface, and can deploy his algorithm to the OSGi-based existing software.

4.4 Medical Imaging

A medical imaging related research group or company wants to create an application which integrates different components to achieve a complex work-flow. This work-flow includes preprocessing of image data on a server, the retrieval of the preprocessed data from the server, the loading of the data into memory, suitable visualization of the data, specific user interaction, followed by the calculation of some parameters and storing them in a database. Because the computational complexity of some of these components is very high and the execution time of the complete work-flow is critical (cost factor), the components are written in C++ and highly optimized. Ideally, each component would communicated with the others through well-defined interfaces implemented as OSGi services to increase their reusability and lower update and maintenance efforts for deployed systems.

5 Requirements

5.1 OSGi Compatibility

N-OSGi-1 Native OSGi MUST use a well defined bundle format similar to the format described in the chosen specification on [5]. This format will include specific Native OSGi headers

N-OSGi-2 The ZIP format MUST be used for bundles

N-OSGi-3 For Native-OSGi a specific header prefix MUST be used

N-OSGi-4 Native OSGi MUST follow the life cycle for bundles as defined in the specification on [5].

N-OSGi-5 Native OSGi MUST detail a service layer and registry similar/equal to the chosen specification. This support SHOULD be based on the service layer described in the chosen specification on [5].

N-OSGi-6 C services MUST be represented using Structs with function pointers.

N-OSGi-7 C++ services MUST be represented using Interfaces.

N-OSGi-8 The service registry SHOULD be able to handle C as well as C++ services.

N-OSGi-9 Native OSGi MUST detail how the Module Layer should be supported. This support SHOULD be based on the module layer described in the chosen specification on [5].

N-OSGi-10 The module layer MUST support multiple libraries.

N-OSGi-11 The module layer MUST support export control of libraries through the use of export/import headers in the bundle manifest.

N-OSGi-12 The module layer MUST support versioning using export versions and import version ranges.

N-OSGi-13 Requirement 1 – 12 SHOULD be based on the OSGi Core Specification. Any version equal to or higher than 4.2 can be used [5].

5.2 Native OSGi

N-OSGi-14 Native OSGi MUST support C and C++. This support has to be in language natural matter. C++ developers should be able to use C++ constructs, whereas C developers should be able to do the same using C constructs.

N-OSGi-15 Native OSGi MUST support consumption of C services in a C++ Framework implementation and vice versa. Usage of these services MUST follow N-OSGi-6 and should be solved in a transparent way without any additional work on the consuming end.

N-OSGi-16 Native OSGi SHOULD introduce no or little overhead when issuing calls between bundles. Where possible wrappers must be avoided. The case of C – C++ interaction will be an exception to this requirement.

N-OSGi-17 Native OSGi MUST NOT rely on functions of specific operating systems NOR alter the standard system environment (e.g. standard runtime libraries)

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

[3]. Sensor Technology Applied in Reconfigurable Systems, http://starsproject.nl/

[4]. S. Kächele, J. Domaschka, H. Schmidt, and F.J. Hauck, nOSGi: a posix-compliant native OSGi framework. In Proceedings of COMSWARE. 2011, 4-4.

[5]. OSGi Alliance Core Specifications, http://www.osgi.org/Specifications/HomePage

6.2 Author's Address

Name	Alexander Broekhuis
Company	Luminis BV
Address	IJsselburcht 3 6825 BS Arnhem
Voice	+31 622393303
e-mail	alexander.broekhuis@luminis.eu

Name	Steffen Kächele
Company	University of Ulm
Address	Albert-Einstein-Allee 11 89081 Ulm
Voice	
e-mail	steffen.kaechele@uni-ulm.de

Name	Sascha Zelzer
Company	German Cancer Research Center
Address	Im Neuenheimer Feld 280 69120 Heidelberg
Voice	
e-mail	s.zelzer@dkfz-heidelberg.de

6.3 End of Document