RFP 0160 Application Framework

Draft

22 Pages

Abstract

This RFP gathers the requirements for the development of an application framework for OSGi, working title: OSGi en Route. This framework encompasses an architecture, IDE templates, an IDE/command line tool, handbook, tool chain, and best practices. The purpose of this framework is to provide a significantly easier starting platform for new users. By providing a complete concrete solution, including toolchain, instead of the modular approach of the specifications, it is expected that adoption will significantly increase.

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	02/06/13	Peter Kriens, OSGi Alliance, Peter.Kriens@aQute.biz
	19/08/13	After review from BJ

1 Introduction

After 15 years of focusing on specifications, the OSGi Alliance has an impressive array of a core framework and component specifications. These specifications have found traction in high end products like embedded, application servers and by many early adopters. However, OSGi never took off as one could have expected looking at the breadth of the technology and its corporate backers. Part of the hesitancy was caused by the threat of another module system in Java 7, which is fortunately gone now for the foreseeable future (moved after Java 8), but it is clear that mainstream adoption has been laggard so far.

There are a number of reasons for the lukewarm reception, these reasons will be discussed in this document's problem section. However, the summary is the lack of a complete example that leverages OSGi to the hilt and does not try to push a square peg through a round hole, i.e. apply a previous model onto OSGi like for example Eclipse and OSGi JEE servers. Though increasingly successful there are no real life examples of 100% µservice based open source applications.

During the development of a reasonably sized application (https://jpm4j.org) by the author of this RFP it became clear that there are a number of steep thresholds to climb for application developers. Not only are there crucial parts missing, a manual is also absent. Though the specification is very detailed, it is written for implementers of the µservices, not the application developers.

It is therefore paramount that the OSGi Alliance takes a step back and looks at the overall picture. In *Crossing the Chasm*, Geoffrey Moore makes it clear that the mainstream customers are interested in *whole solutions*, not partial solutions. The OSGi Alliance, even after 15 years, shares many of the characteristics of the start-ups discussed in this book; and modularity is the ultimate partial solution. Since the OSGi Alliance's raison d'être is to improve software development by enabling a component market, it is necessary to look at what the *whole solution* actually means.

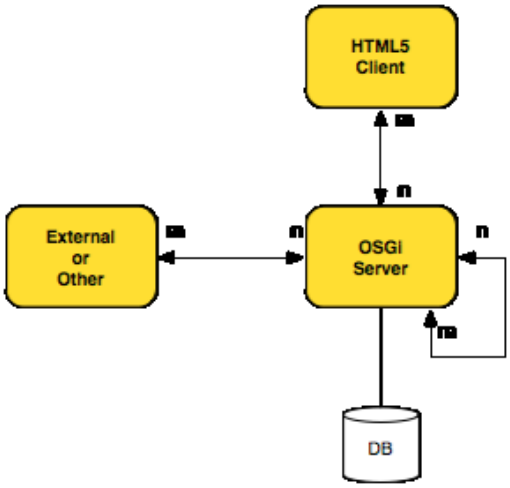
This RFP therefore investigates the requirements around an OSGi Application Framework for web development. It is intended to lead to a number of RFCs that will then result in a set of tools, rules, example apps, tutorials, and a handbook.

2 Application Domain

The following sections define the scope of this RFP and then describe the technologies used to implement these type of applications today.

2.1 Scope

OSGi is used from tiny embedded devices all the way up to mainframes. Its application domain is therefore without bounds. However, this is an unworkable large scope. In the remainder of this document, it is assumed that an *application* therefore consists of a HTML5 browser front-end backed by a one or more OSGi server back-ends. The front-end provides the user interaction while the back-end provides the actual functionality/business logic, data persistence, and collaboration with other servers. The archetypical architecture is depicted in the following figure:



The following sections enumerate the important subject areas and provides a summary of how they are developed. Since this is a very rich area, it is kept short to point at the relevant areas and define a vocabulary to discuss these later in the problem section. A google search will in general provide more background.

2.2 Basics

This section discusses the foundational technologies to construct an application and thereby define the needed terminology.

2.2.1 Component Interconnect Mechanism

Large applications need to be composed of components, preferably reusable components. In general, components are type coupled to an interface which then leaves the problem of how to find an implementation for that interface. Java *factories* are the de-facto standard interconnect mechanism. Factories provide a standard facade type which can then be used to provide specific implementations with under the cover techniques. A client can thus be configured to run with different implementations depending on these techniques.

An alternative technique, *Dependency Injection*, was made very popular in Java with *Spring* after 2002. In this model an XML file and annotations parameterize the client's objects with instances and configured objects by setting fields and calling setter methods. Since then a number of other libraries have sprung up, like Guice, but Spring is by far the most popular mostly because they also provide a lot of compatible libraries.

Both factories and DI use *dynamic class loading*. In dynamic class loading a class is loaded by its name. In general, most libraries assume there is a single Java *namespace*. Since the Java VM has multiple class spaces, each being its own unique namespace, dynamic class loading can fail and therefore requires special hacks like the Thread Context class loader to provide libraries access to otherwise unreachable class spaces.

Spring's technology was made available in OSGi via Blueprint. Blueprint also used dynamic class loading but is aware of bundle boundaries. Declarative services is a lighter weight form of Blueprint that is native to the original OSGi µservices model.

2.2.2 Configuration

There are few general ways to configure an application outside OSGi. Spring provides some basic configuration through its wiring XML but that option should in general not be used for volatile or sensitive information like database urls, passwords, etc. For this reason, most large libraries provide their own XML or properties files with varying syntax.

In OSGi Configuration Admin is provided for configuration. Since this service has a powerful lightweight API and was available from the beginning it is used extensively in the OSGi world.

2.2.3 Dependencies

Currently the most popular dependency model for Java is M*aven*. Maven describes a source software project with a Project Object Model (POM). The POM uses a transitive dependency model to construct the application's build path, test path, and runtime path. Since these dependencies are transitive, a large number of dependencies result for application level projects.

Since dependencies are transitive, it is possible to require the same library in multiple versions. In single class space environments this results that only the first library encountered is used, potentially allowing the wrong version to be used.

In OSGi, each artifact is self described by its metadata. Dependencies are in general resolved, which results in fewer version clashes when used with version ranges. In runtime, OSGi is the only environment that can actually handle version clashes by running the same program in different versions since each bundle has its own class loader.

2.3 Web Interface

2.3.1 Http Service

The initial 1.0 OSGi release had an Http Service included. A bundle could get the Http Service and register a *servlet* at a given url prefix. Http requests were then forwarded to these servlets. A Http Context could be registered for static resources. Apache Felix added whiteboard support for servlets as well as Filters. These patterns have been adopted in the update for the Http Service which was long overdue.

2.3.2 WAR/WAB Solutions

A developer that wants to build an archetypical system today has a number of options. The most often used route is to create a *Web Archive* (WAR), a JEE standard. WARs provide the standard layout of JEE Application servers. They carry static resources like javascript, html, css, etc. as well as Java classes and the libraries that they depend on. Since the Enterprise Release, OSGi supports this standard. A WAR that can act as an OSGi bundle is called a Web Archive Bundle (WAB). WABs can import packages and require other bundles, they therefore do not have to carry all their dependencies in the lib directory.

2.3.3 HTML Generation

Most application frameworks in Java generate the HTML in the server. For example: JSP, JSF, Wicket, Struts, Spring MVC, Vaadin, etc. These frameworks place a significant demand on server resources since they must maintain all state for all users. This is especially problematic in a cluster based system where state needs to be distributed among the members.

Over the last few years more and more application code has been offloaded to Javascript since HTML5 made it viable for quite large sized applications because most of the portability and runtime performance problems were solved. A number of frameworks, Angular, GWT, JQuery, Dojo, etc. provide application frameworks in Javascript. They do most of the user interaction in the browser and use the server with XML Http Requests with either XML or JSON. These Javascript environments have become viable for full blown user interaction oriented applications.

2.3.4 XML Http Requests

XML Http Requests are used by web clients to retrieve information from the server. Though they were originally designed for XML, they are today mainly used with JSON data. The main protocol uses is Representational state transfer (REST), however, JSON RPC is becoming more popular since REST does not work well when there is a need for more verbs than the few foundation verbs provided by REST. For systems that split the application between a Javascript front end and a Java backend it is often hard to represent each request in the REST model.

On the server side, the Jersey library is very popular for REST although there is an impedance mismatch with OSGi because the library uses dynamic class loading and works therefore not elegantly with µservices.

2.3.5 Transport Optimizations

Caching, compression, and multi-file downloads are crucial for high performance websites. Caching requires headers to be set on HTTP requests, compression requires the manipulation of headers and compression of the streams. Multi file downloads allow less connections. New developments, like the SPDY/Http 2.0 protocol, etc. provide more optimizations. These optimizations are in general not concerning the application developers but are crucial for providing a fast web experience.

2.4 Server

This section discusses the common most important functions needed by a web application server.

2.4.1 Security

Though OSGi has User Admin, this is in general not a sufficient solution for modern web based applications. In practice, people use Spring Security and ad hoc mechanisms for authentication, authorization and identity management.

2.4.2 Persistence

Persistence in Java is mainly relational (though all NoSQL databases have Java drivers out of the box). Relational databases are abstracted with JDBC, which is widely supported. There is an OSGi standard for JDBC drivers to register a DataSourceFactory service, this is not widely supported by database vendors yet.

On top of JDBC there is Java Data Objects (JDO, older) and Java Persistence Architecture (JPA, newer). JDBC and JPA are standardized in the OSGi Enterprise Edition, while JDO is not. However, Google App Engine selected JDO as its API for Google Big Table since JDO, in contrast with JPA, allows non-relational databases. This feature has made it more attractive since the advent of NoSQL databases.

In general, JPA is seen as the 'standard' Java Persistence Architecture.

2.4.3 Transactions

In JEE transactions are as much as possible hidden from application code. Applications retrieve 'transactioned' objects from JNDI. In OSGi, the JTA specification mandates the registration of a Transaction Manager, which allows any component to participate in transactions.

2.4.4 Queueing

A queue provides support for initiating a task that is, guaranteed in certain cases, to be executed later. This allows the caller to continue while the task has not yet been executed. Queuing in JEE is handled through JMS.

There is no dedicated OSGi queueing standard. In general persistent queues are used from popular open source libraries, with or without Java Messaging System.

2.4.5 Batch Jobs

In every non-toy application there is a need to execute function on a given time or when a certain condition occurs. The most popular package is Quartz but Spring Batch also provides this function. In JEE 7, JSR 352 specifies a batch model. This model heavily relies on XML and dynamic class loading. It also specifies a Batchlet, an interface for running and canceling batch jobs. However, a Batchlet is virtually identical to a Callable.

2.4.6 Search

Virtually all applications have free text search functionality. Free text does not work well with most standard databases and is therefore usually handled with another product like for example Elastic Search or Apache Lucene.

2.5 Application Frameworks

Outside the Java world there are number of highly successful frameworks that took the whole solution approach. The archetypical one being Ruby On Rails; in the Java/Scala world the Play framework is making inroads. These frameworks are holistic in that they are not about the parts but make it possible for the application developer to understand and manage the forest instead of getting lost among the trees. These frameworks consist in tools, rules, and documentation.

An application framework provides a tool to create and extend an application. In general, the tool maintains the domain model. The domain model consists of the data entities and their behaviors. Knowing the domain model, the tool can provide automatic scaffolding, enabling users to modify working code instead of having to create new things. All models use the Model-View-Controller paradigm, where the tool can create new models, viewers, and controllers. The tool then maintains or ensures the relations between these parts.

Plugin support in application frameworks provide additional capabilities.

2.5.1 Ruby On Rails

The archetypical application framework is Ruby-on-Rails (RoR). It relies on Ruby, a powerful dynamically typed language. The tool is a command line app, rails, that contains hundreds of commands. A command like `rails new my-app` creates a directory and in that directory provides a complete setup for a rails application, this application has scaffolds/defaults so that the application can immediately be run, also via rails command.

Additional commands then can add domain entities, views, controllers, etc. When the application is finished, it can be packed and deployed locally or in the cloud. A very popular place to deploy is Heroku, this deployment is fully integrated with the RoR life cycle.

2.5.2 Grails

Grails is an application framework based on Groovy, inspired very much by Ruby On Rails. Its structure and files are very similar to RoR but Grails is based on Groovy instead of Ruby. Grails is quite mature because it is built upon Spring.

Grails is supported by the SpringSource Tool Suite.

2.5.3 Play Framework

A more recent entry in the market is the Play framework. The play framework originally used Scala as the underlying language but it also allows Java or a mix of these languages. It provides an efficient actor based programming model.

2.5.4 Google App Engine

The Google App Engine is an (almost) Java based environment with an extensive API for web applications. Though Google does not have a command like RoR, it does provide an Eclipse plugin that provides support for using its API. Jboss, with Cape Dwarf, is implementing the whole GAE API in open source, allowing the GAE applications to also run outside Google's data centers.

2.6 Other Languages

No bundle is an island. A modern web application consists of many different files written in many different languages. The following are of resource types present in almost any web application:

- Javascript (or derivatives like Coffeescript, Dart, etc.)
- Cascading Style Sheets (CSS) – Provide the styling/theming of a web site. Since CSS is very redundant, a number of languages like less and scass provide a similar syntax but have variables, imperative capabilities, and nesting.
- Hyper Text Markup Language (HTML)

Even though a web application consists of many different resource types, they in general refer to each others name spaces. That is, names from one resource type can be used in another resource types. Renaming an entity can require editing many different files if the name type is not supported by the refactoring engine.

A web site has a single namespace for its web resources, this is sometimes hard to manage with components that each require a place for shared an their private resources in this namespace.

2.7 Tools

Developing is not about language alone (although sometimes one gets the impressions from discussions on the net). The whole toolchain is of critical importance, a good build system works as a multiplier for a development team.

Build tools automate the process of building the final artifacts that will be deployed. They often provide a plugin and/or scripting model so that user defined actions can be integrated in the build.

2.7.1 Source Control

Source control is ensuring the archiving of source code, restore prior versions, as well as providing separate development paths. Currently git is the most popular tool. There are several very popular hosting sites for git with github being the most widely known. These hosting sites also provide documentation and issue management.

2.7.2 Issue Management

Issue Management allows people to report bugs, request features, and synchronize different development paths. Very popular is JIRA, however, Github provides a simple issue tracker that becomes quite powerful due to its integration with git commits.

2.7.3 Integrated Development Environments

Java development is blessed with a number of Integrated Development Environments (IDE) among which Eclipse, Netbeans and Intellij. The environments allow developers to work extremely interactive, whenever a change is made, any dependents are updated. IDEs also provide automation, navigation, and refactoring. Since Java is a statically typed language, the IDEs can provide impressive support in completion and refactoring that is absent for most other languages.

A number of developers prefer to work outside IDEs, using simple editors and build tools. These people feel that IDEs put too many layers between them and the running code. These intermediates can make debugging quite complex since there is so much code involved that is not of the authors.

Eclipse has two streams for OSGi development. The original development plugin was PDE. It is mainly geared towards making Eclipse Plugins, which are also bundles. Recently the bndtools Eclipse plugin provides an OSGi development model that is much more geared to plain OSGi.

2.7.4 Build tools

The archetypical build tool is make. Make provided a structure for shell scripts and combined it with dependencies. Make was replaced in the Java world by ant, where the ant files became xml. The primary reason for this move was that ant was portable because it was written in java.

This portability has been a long term driver for Java build tools. Though it is desirable, it implies that all tools must be written in Java it is undeniable that most tools are **not** written in Java. Especially in the web world tools written in Javascript based on node.js have become quite popular.

Maven is currently the primary build tool in the java world. Its model is different that it provides much more structure but therefore also makes it harder to to do unforeseen things. A crucial aspect of the maven eco system are maven repositories. These repositories are wildly successful in other build tools.

For OSGi development, maven uses the bnd library. This same library is used in bndtools. The library provides an abstract bundle build system that can be leveraged, and is leveraged, in many build tools.

Most new languages, Scala, Groovy, provide their own build tools. Scala provides sbt and Groovy provides gant and gradle. All these tools use bnd for the OSGi manifest generation.

Build tools build a set of sources and creates deployment artifacts. Depending on the deployment model, such an artifact can be a bundle, a WAR file (can be deployed on an App server, contains all its dependencies), or an executable JAR (a JAR that contains all its dependencies and specifies a Main-Class in its manifest.

2.7.5 Continuous Integration

Builds are usually optimized so that rebuilds are faster than the first run. This optimization can create a path dependencies: the build runs on developer A's machine but not on developer B. Best practice today is therefore to provide a continuous integration environment where the code is checked out from the source control system and then build from scratch. After the build the binary artifacts can then be placed in a repository.

2.7.6 Release Management

Release management is the process of assigning version numbers and making build artifacts more widely available, first for Quality Assurance (QA), and later for production or other developers.

Maven provides a generally felt awkward release process. In OSGi, release management is quite acute. For this reason, bnd(tools) provide extensive tooling around this.

2.8 Terminology + Abbreviations

- JSE – Java Standard Edition
- JEE – Java Enterprise Edition
- GWT – Google Web Toolkit
- Executable JAR – A JAR with all its dependencies inside the JAR and a Main-Class manifest header that points to the JAR's main-class.
- µService – An OSGi service.
- DI – Dependency Injection (Spring, Guice)

3 Problem Description

3.1 General

OSGi provides a generally acclaimed component system that enforces modularity by linking components through distinct ports, a.k.a. µservices. This model is superior to any alternatives, however, it is also *innovative* which implies that there is little prior art. In a new market this would not be cause of concern since the always present early adopters would quickly establish state of the art applications and libraries. However, when OSGi became viable for larger adoption in the enterprise software world it ran into a huge wall of existing free open source Java software that was incompatible with OSGi in several ways.

Unfortunately, the hack of dynamic class loading used to interconnect modules with the mainstream DI is fundamentally not modular, making it run into the modularity fences of OSGi. This makes running existing open source projects on OSGi a painful exercise (as can be witnessed on Stack Exchange). This extra work required to use OSGi, even though it clearly improves the application's code base, is often deemed too much work and people remain in JSE.

Another problem was that most popular open source projects are rarely designed as reusable components. This means that many projects go out of their way to provide a complete environment instead of providing a cohesive component. The nature of open source, contributions from everywhere makes most open source projects highly coupled to other projects.

Analyzing popular open source products make it clear that many POMs are not completely correctly identifying dependencies. However, Java is quite robust and often code that would blow up when executed is never reached. Alas, in OSGi these undefined couplings are detected before the application is started and then refused to run, which is a bit of a showstopper; developers then tend to blame the messenger giving OSGi a bad name.

A less clear problem with OSGi/Java is the flexibility paradox that *there is just too much choice*. As a RoR developers states: *"Imagine asking five experienced Java developers to design a structure for a new Java web application. If you had anything less than five spectacularly different results I would be shocked. This is not so with Rails."*[1] Java environments tend to focus heavily on configuring and less on relying on defaults. RoR explicitly promotes convention over configuration providing clear design paths. Though flexibility is in general deemed better, it can be the enemy of good when it introduces too much complexity. This is the case in Java.

Therefore, by *doing things right* OSGi made it very painful to port existing applications and libraries since it required fixes to a 'working' code base up-front. If OSGi had been accepted much earlier, it would likely have been adopted widely since doing it right, the OSGi way, is not more work than doing it wrong. However, the JSR 277 ruckus (Java modularity/Jigsaw) created a wait and see attitude in the industry while at the same time open source projects solidified the bad practices.

That said, the OSGi Alliance was not without blame. Even before the OSGi Alliance was founded the complaint that OSGi was a beautiful set of specifications but it lacked a manual and toolchain. An Ericsson employee never forgot to ask the author of this RFP at each visit: *"Yes, beautiful! But how do we put it all together?"* In our quest to develop specifications we ignored the need for more guidance on the holistic level. When this issue was discussed early in the life of the Alliance the board decided to not take any initiatives in this area since this was the realm of the members; supporting customers this way was deemed a business opportunity. And to a certain extent this was a correct decision; ProSyst clearly supports their embedded market with a range of products and trainings, just as IBM supports their customers with Rational products.

However, these commercial products have no impact in the open source community. This would not be an issue if the open source community had not become the fertile feeding grounds for enterprise software development. Trends in the open source community are more and more adopted in commercial software development for obvious reasons: access to high quality components and trained engineers. To have success in the commercial software shops it is crucial to have traction in the open source world.

The OSGi framework is a perfect foundation for Application frameworks like RoR and Play. If there had been more bundles that really leveraged OSGi then such a framework would already have been created by the open source community. Unfortunately, this has not happened.

3.2 Goal

The goal this RFP is seeking to address is simple: increased adoption. Adoption is lagging likely because it is too hard to get started with OSGi. Despite significant improvements in tooling and libraries, getting a simple web application up and running is a non-trivial exercise. First there is just too much choice. Starting an OSGi application requires a large number of choices up-front; for new comers it is hard to judge what the impact is of all those choices. What framework, what services to use, what bundles to use, etc. Many of those choices can have far ranging implications that are hard to oversee initially.

What worsens the situation is that many popular open source projects are awkward in an OSGi setting. Many projects link components in ways that are not modular and therefore not compatible with OSGi. OSGi also is much more strict than traditional Java, making it often the messenger that developers like to shoot.

Another problem is the lack of examples that leverage OSGi to the hilt. OSGi's Declarative Service provides a very powerful programming model since it provides a defined life cycle with Configuration Admin support. This model is much more powerful than most, especially long time OSGi developers, think. Unfortunately, few bundles are available in open source that really use it.

1 See also http://discuss.joelonsoftware.com/?joel.3.219431.12 for a more comic illustration

What is therefore needed is an easy way to build an OSGi application that leverages OSGi instead of fighting it. The solution should address the whole OSGi development life cycle. What is envisioned is a tool like RoR that allows developers to create applications and extend these applications over time.

3.3 Phasing

This RFP has a very wide scope and it is not feasible to execute this at once. It is foreseen that it will be executed as a number of RFCs over a number of iterations.

- Technology selection — Establish a common set of core technologies in conjunction with the OSGi communities.
- Drive Technology — Work with the industry and open source communities to provide implementations that conform to the selected technology
- Tool Chain — Establish a tool chain in conjunction with the OSGi communities
- Tool — Create a tool to automate many of the steps necessary to setup an OSGi project
- Vendor Support — Work with service companies like Github, Cloudbees, etc. to setup an integrated environment.

4 Use Cases

Use case Q and P are about how companies work today to sketch the market. The use case is based on some interviews with actual developers and managers. Use case R is constructed from several experience reports about Ruby on Rails. Use case F describes how a future company could start with the results of this RFP: Tool X.

4.1 Company Q

Company Q is a startup that started 3.5 years ago. It provides a data mining tool for a very high end production process that generates terabytes of data per week. The data is retrieved from the production line in a constant process and integrated with a NoSQL Cassandra database. The server filters and normalizes the data and calculates aggregates and analysis reports. The data can then be navigated in numerous ways. The user interface, written in GWT, is highly geared towards interactive charts visualizing the underlying production data. They handle security through Spring and an LDAP server. Their key selling point is the domain knowledge about the process that is encoded in the software combined with very fast response times. The server is based on Spring components, the language used is Java.

Q has about 8 developers, a scrum master, 3 testers, 5 domain experts, and management. They have two offices in France and are expanding to the US to support potential US customers. The follow an agile process with sprints of 6 weeks. They are ISO 9001 Quality Management certified and work on ISO 27000 Information Security.

The input of the sprint is a specification document that is presented during a day long session to the team. A week is used to come up with an agreed design plan. Planning is quite accurate, between 10-30% up/down. Management expressed great interest in the quality of the planning (both up and down!) and sees the duration secondary.

Developers take parts or are assigned part of the plan and then code and test it. All developers use in a single workspace that is checked out as a whole in Intellij. Since all cross dependencies are in the workspace and there are no external dependencies it was not deemed necessary to maintain versions. External dependencies are consumed through a Nexus server, the estimate of external dependencies was said to be 6000 (which seems rather high). Adding external dependencies requires a management decision. They actually monitor third party transitive dependency downloads since they discovered a problem with a deep transitive dependency that had the wrong license.

After the workspace is committed to SVN, a build is kicked off in Jenkins. Jenkins builds and then stores the artifacts on a public file system, from where they are consumed by QA for manual testing and automatic integration testing as well as delivery to customers.

Deployments are handled with Chef or Puppet. They have a small number of processes: importing, analysis, and web serving. Externally they run Cassandra. The processes are connected through a message queue.

Problems are reported in Jira and assigned team members. Official deliveries are made by tagging SVN and creating scripts that take the Jenkins artifacts. The have a number of trials that use the system as Software as a Service (SAAS). They rent space in a datacenter for this.

Overall the company uses a classic Spring/Maven development process typical for enterprise applications. They are happy with Spring and to a lesser extent Maven, though they still regard maven as the best option, some investigation is going on into Gradle.

The company is still in the start up phase and has no commercial customers yet. The software is installed in a few locations for evaluation. This of course means that they have the advantage of a new codebase without any dependencies on it, nor have they been tormented with a customer specific patches. This is likely the reason they do not yet have a powerful need for modularity.

4.2 Company P

Company P is a small 3 men company focusing on web services on the IBM AS 400. They have a cash cow product that converts text based invoices into PDF based invoices but recently started a new project.

They use PHP, not Java. 12 years ago it had sent its engineers to a Java course at IBM but when they came back they were utterly confused with Java and its objects and the company switched to PHP. They prefer to code in a procedural style and think objects are complex in general. Their tooling is:

- AS 400
- Eclipse with the Zend IDE
- SVN

4.3 Company R

Company R is a heavy Ruby-on-Rails shop. It has 6 developers working full time over a number of years on a relatively complex rails system. The system consists of a number of (relatively) independent Rails apps and gems (Ruby modules). The application's are scaled by instantiating multiple servers, this is relatively easy in Ruby since

most state is maintained in the request. The system consists of a large number of database tables (>200). Since these tables tend to come in clusters, they are assigned to specific applications that then acts as a facade to the tables for other servers, either through REST or via read only access. Static pages are handled through an Apache web server since serving this from Ruby is excruciatingly slow.

Developers work on applications that can run against a development system. Code is stored on Github repositories. To keep code quality high the team uses code reviews. Code reviews are necessary to prevent people from becoming 'clever', to enforce common coding standards, minimize the effect of typos, and keep the complexity of the code base in check. The team follows the RoR 'convention over configuration' mantra, which makes it easy for new developers to get up to speed and minimizes discussions around designs. Despite this, sometimes productivity slows to a crawl and it becomes necessary to refactor the code base.

The company provides has very strict policies for dependency management. This is mostly to prevent version clashes since Ruby code is not very backward compatible; it is necessary to strictly version the gems, apps, and interpreter. They use the Ruby Version Manager that handles the differently Ruby interpreter versions that are needed on different systems.

Issues (bugs and requests for new functions) are handled through the Github issue tracker. The project manager triages the bugs and assigns them to the appropriate persons. For new features, Company R uses Cucumber, which is based on Behavior Driven Development (BDD). Customers and product management writes *feature files*, describing use cases in a *structured* free-text form[2].

Company R uses follows the Agile movement with sprints of 4 weeks and one week cleanup/planning. In the cleanup phase problems are analyzed. The features are used as tasks in the planning game. The selected features are then given to developers for implementation.

The developers implement the feature and write unit tests for this features. Ruby test cases can be linked to the feature using regular expression patterns. Company R standardized on RubyMine[3] as the IDE, which supports this workflow.

After all unit tests are running they commit their code to Github. Submitting to Github triggers a Continuous Integration server that will then build the code and run its tests, informing the team of any failures.

Code can be deployed into production directly from the Github repository to Heroku, a Ruby hosting provider, by creating a 'heroku' branch and designating this branch for deployment with the Heroku command line app. Every time the branch is committed, the app is deployed and the servers restarted. Static resources for the Apache Web servers are deployed in a similar, but proprietary, way.

Since Ruby is very version-sensitive, every 6 months one sprint is dedicated to moving to the latest versions of the used dependencies and, if necessary, a new Ruby interpreter.

4.3.1 Company F

Two years from now, company F must deliver a new application with a web front end, an application that will be used by postal workers for delivering packages. After a selection they choose X, a tool to simplify OSGi application development. X encodes best practices of OSGi web applications development and heavily relies on convention over configuration and thus to a certain extent simple over flexible. Tool X defines a strict project layout and configures other tools like bndtools, maven, to work with this layout. X can be used as a command line tool or as an Eclipse plugin but company F decides to use the Eclipse plugin. This plugin is installed in the latest Eclipse via the Eclipse market place.

2 Cucumber Behavior Driven Development: http://cukes.info/
3 RubyMine: http://www.jetbrains.com/ruby/

New developers can follow a tutorial on the X website. This tutorial shows in detail how X can be used to build web applications and shows how the different options can be used. Tutorials use Eclipse's facilities to make them interactive.

X provides a wizard to create a new application. The wizard will ask a number of questions to create a project of the desired type, populating it with compile and runtime dependencies. The wizard provides the selection of the authentication/authorization model, database type, etc. Once the application is created it provides a skeleton app that can be run immediately, allowing for incremental development. In this application, scaffolds are provided for defined data models.

An application is an Eclipse project with a layout that provides for all the aspects that make up a web application: server code, client code, css, html, data model, etc. It has selected tools to simplify development, e.g. less[4] is provided to simplify CSS development. Applications are relatively small because all functionality is provided as bundles.

X takes advantage of OSGi's strong modularity, it provides extensive guidelines for naming the resources used in web applications to prevent name clashes between modules and promote collaboration. For example, it recommends to use the Twitter Bootstrap CSS class names for all web development and makes sure that none of its components can clash with this namespace. It also provides recommended bundle layouts so that bundles can be used to provide resources like HTML, images, etc. without conflicting with other bundles. This standard layout is then used to support web request caching strategies.

Modularity is the technique to break a problem in distinct parts; these parts can then be shared between applications. For this reason, X recognizes *applications* and *libraries*. Applications are not reusable and can therefore freely depend on any library without increasing transitive dependencies. Libraries are the reusable components and must therefore be designed with care to minimize these transitive dependencies. X provides also wizards for libraries and tools to analyze dependency graphs.

X has the concept of a workspace. A workspace is a cohesive set of related projects (applications and libraries) that form a single unit for source control, testing, continuous integration, and release. Dependencies inside the workspace are maintained together, other dependencies are maintained through a server based on the OSGi Capability/Requirement specifications. This server holds all of Maven central and provides scripts to generate bundles of the latest Javascript libraries, which can easily be shared with others.

X is based on the HTML5 model: The client code is based on 'single-page' Javascript applications that communicates with a Java based server over REST and JSON RPC. Bundles can provide both server and client side resources. X provides a comprehensive testing framework for both the Java as well as the Javascript code, allowing Javascript tests to run inside multiple browsers.

Bugs and new features are managed through Mylyn with a Github backend. Company F uses agile as its development model. It uses Cucumber to provide the feature descriptions, these descriptions are linked in Eclipse to related Java(script) code, data models, test cases using Mylyn[5], Wizards are provided by X to create features and related resources like code, css, data models, etc. Features are selected in a planning game and used to drive a 4-6 week sprint with one week of analysis and planning.

All bundles and exported packages are versioned and baselined (compared for compatibility with a previous release) inside the Eclipse IDE, imports are semantic versioned based on the version of exported packages on the build path.

X sets up Cloudbees, a CI provider, to automatically build after a git push. After a successful build (which includes semantic version checks) it will provide the bundles to Quality Assurance (QA). If QA approves the bundles, they become available for deployment to production on Amazon EC2 through a wizard in X.

4 Less: http://lesscss.org/
5 Mylyn: http://www.eclipse.org/mylyn/

5 Requirements

5.1 Goal

- Reduce the threshold for new users to adopt OSGi for Web applications by providing an OSGi application framework that significantly simplifies OSGi development.

5.1.1 Deliverables

- An IDE based tool to create new projects and maintain these projects.
- Provide a toolchain that includes source control, IDE, offline build, issue management, continuous integration, and release management.
- Establish OSGi best practices document in the context of the application framework.
- Provide an open catalog of open source and commercial components that are compatible, as stated by the vendor, with the application framework.
- Provide an extensive handbook for the application framework.
- Provide a demonstrator application.
- Provide tutorials for using the application framework.

5.2 Application Framework Architecture

- µservice based
- Base the Application framework on HTML5 with a Javascript GUI inside the browser and the server serving JSON packets.
- Use JSON RPC for the communication layer through a service API
- Support REST for external access through a service API
- Support Servlets using the whiteboard approach
- Use JPA for the persistence layer
- Provide static pages in bundles inclusive caching, if- headers, compression, and ranges
- Modularize Javascript/CSS/html support with OSGi (e.g. can deliver bootstrap as a bundle)
- Provide naming scheme for web resources to prevent clashes in a component based application.
- Provide security model

- User management
- Authentication
- Authorization
- Web attacks like cross site scripting, etc.

5.3 IDE Based Application Tool

- Create/Delete projects based on templates and wizards
- Maintain a central data model that can be used to derive persistence models, validation (also in browser's javascript), can drive viewer/controller scaffolding, marshaling and other use.
- Provide CRUD operations on the central data model
- Allow operations to be done through a GUI and command line
- Provide the capability to use the tool library in different IDEs
- Allow applications to be packaged as executable jars.

5.4 Toolchain

To address the large number of choices that a Java developer is facing, the tool must provide an-out-of the box experience. This will include the choice for actual products since offering another toolbox with a myriad of choices forfeits the purpose of this exercise. However, it is important that the tool is based with an open API and that the chosen tools can be replaced. This RFP does not prescribe any particular technology and over time a specific tool can also be replaced by another. However, actual RFCs for this RFP must make actual choices, likely the most promising tool at the time of choice.

- Provide simple access to a full example tool chain for
  - Source Control
  - Issue Management
  - IDE
  - Command line build that is compatible with the IDE
  - Continuous Integration
  - OSGi Management + Diagnostics (e.g. Webconsole)
  - Application packaging (delivering an executable JAR)
  - Release management
- Make it possible to substitute parts of the toolchain with different tools by providing an open API

5.5 Best Practices

- Develop a concise set of best practices around
  - Bundle packaging and headers
  - Architecture
  - Building
  - Versioning/Release Management
- Develop a tool to verify that a bundle follows applicable practices
- Provide branding of bundles following these best practices in collaboration with marketing
- Leverage the experience of the EGs by actively seeking advice and feedback

5.6 Component Catalog

- Provide a web based component catalog of components that are compatible with the application framework.
- Allow self management (stage components, release new components, withdraw components) by vendors/communities
- Provide access via the OSGi Repository service

5.7 Handbook

- Provide a handbook that describes the application framework, its development process, and use of associated technologies.
- Provide how's, but be light on details (these can always be found in manuals of the different tools)
- Provide extensive why's
- Provide examples and tutorials
- Provide an ebook, HTML5, and PDF version
- Underestimate the developer's knowledge

5.8 Demonstrator

- Provide a demonstrator Blog application
  - Multiple users
  - Free viewing

  - Logged in users can comment, post blogs, edit/delete their own blogs/comments
  - Free text search
  - Live update when a post changes (demonstrate Server Side events)
  - Email notifications for comments and posts
- Excellent GUI
- Demonstrate customized Bootstrap delivered through a bundle
- Extensively commented and documented source code
- Demonstrate pluggable providers for database, etc.
- Demonstrate translations in different languages
- Provide a tutorial building this app

5.9 Non Functional

- OSGi 'correctness' must be chosen over backward compatibility
- Leverage existing specifications
- Develop missing specifications as RFPs/RFCs
- Reference implementations are preferred if all else is equal
- All included components must be licensed under ASL2, EPL, or MIT.
- Referred components that are not included may use other licenses but must make the licensing explicit.
- All work must be based on modular principles and use µservices where applicable.
- Integrate with open source projects and vendors that can provide solutions
- Actively collaborate with open source projects to get their support.
- Prefer providing patches to an open source project for missing functionality over providing this functionality itself.
- Actively seek collaboration with OSGi members, vendors, and volunteers for this work
- Enable vendors like Cloudbees, Github, etc. to create commercial models based on this effort.

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	Peter Kriens
Company	aQute
Address	9c, Avenue St. Drezery, 34160 Beaulieu, FRANCE
Voice	France 0633982260
e-mail	Peter Kriens@aQute.biz

6.3 End of Document