RFP 122 OSGi Resource Repository

Draft

8 Pages

Abstract

As OSGi becomes part of the mainstream, the number of resources that a system has to define and use quickly becomes quite large. After the decision to develop large scale systems as modularized products, the problem of how to organize, categorize, search and define these systems becomes critical. A common repository definition which federates the ever growing number of resources available to the OSGi developer is desired such that it can supply the foundation for sophisticated tooling, integrated into all aspects of the software lifecycle from discovery to development through QA to deployment and maintenance.

0 Document Information

0.1 Table of Contents

0 Document Information....2
0.1 Table of Contents....2
0.2 Terminology and Document Conventions....3
0.3 Revision History....3

1 Introduction....3

2 Application Domain....3
2.1 Terminology + Abbreviations....4

3 Problem Description....4

4 Use Cases....5
4.1 Discovery....5
4.2 Build....5
4.3 Provisioning....5
4.4 Patching....5

5 Requirements....6
5.1 Negative Requirements....6
5.2 Functional....6
5.3 Discovery....6
5.4 Dependency Resolution....7
5.5 Non Functional....7

6 Security Considerations....7

7 Document Support....8
7.1 References....8
7.2 Author’s Address....8
7.3 End of Document....8

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 7.1.

`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	*MAR 14 2009*	*Initial version* *Hal Hildebrand, Oracle Corporation* hal.hildebrand@oracle.com
0.1	*MAR 18 2009*	*Incorporated feedback from conference call* *Hal Hildebrand, Oracle Corporation* hal.hildebrand@oracle.com

1 Introduction

This work is based on the excellent work by Richard Hall with the Oscar Bundle Repository and with the previous version of RFC 112 developed in collaboration with Peter Kriens. As RFC 112 was developed without an RFP, when the RFC 112 was taken back up, it was quickly discovered that the RFP process was critical to getting concensus around the RFC and thus this RFP was born.

2 Application Domain

OSGi specifications are being adopted at an increasing rate in the Java community. With the advent of modularization, the problem quickly becomes how to manage the large numbers of bundles both within any particular project, but across projects, companies and throughout the open source community. Repositories based on Maven and Ivy can provide some measure of support, but do not capture the richness and depth the OSGi dependency model is capable of.

When developing applications, we start from a vast base of open source and private libraries that are available to the project. The dependencies between these libraries quickly grows in complexity and developers need to quickly determine which versions of the required dependencies can be constructed to successfully run in the system being developed. To further complicate matters, a system under development is not static and evolves during its lifecycle and complicated dependency relationships between different versions of the system need to be recorded and use to determine deltas, patches and the further dependency relationships of other systems on the developed modules.

Repositories provide discovery services which allow the developers and planners to browse and navigate these interrelated resources. These discovery services should be easily integrated into the tools used by developers so that they can quickly assemble existing solutions relevant to the problem being solved. These repositories not only span multiple organizations, both internal and external, but span many different mediums from shared discs to read only media to massive database backed content delivery systems.

When determining what to deploy and how to provision a system, we would like to indicate only the critical and defining resources of the system and have the dependencies of this set determined automatically through a resolution process. This transitive closure over required resources should be installable from the information returned from the repository so that we can provision processes and applications with only the resources actually required.

2.1 Terminology + Abbreviations

3 Problem Description

Whenever a problem is solved, the reality is that an additional set of problems are created which didn't exist before the solution solved the problem. As developers adopt OSGi and begin to modularize their existing systems and develop green field systems using OSGi, the explosion of modules, their interrelated dependencies and the solution to constraints of desired systems quickly become hard, if not impossible to manage. The sheer number of bundles available even in the privately controlled repositories of large software development organizations quickly becomes overwhelming. As repositories spring up in public and private institutions, individual developers and organizations want to take advantage of the innovation happening in these other organizations and need a mechanism to federate these repositories into a coherent system which can be searched, queried, annotated and organized. System administrators need sophisticated tools which can quickly tell them with a reasonably degree of accuracy what the impact of the uptake of patch sets will have on existing installations. Without a common metadata facility which organizes the disparate resources into a constantly resolved sets these problems are pushed back to the organizations to develop, hindering the uptake and exploitation of OSGi within their development organizations.

4 Use Cases

4.1 Discovery

A developer on a large project knows that there are resources available which provide the solution for the problem at hand, but the question is where are these resources and perhaps more importantly, what will these resources require as dependencies and whether the available versions compatible with the other constraints in the system.

The developer should have sophisticated tooling which can be configured with the certified repositories the project can draw upon. These tools will provide the developer with an interface which uses the supplied metadata of the repository to guide the developer in browsing the repositories by revealing only the resources which meet the constraints of the project being worked upon. These resources can be filtered by acceptable licenses and execution environment. When the developer finds the desired resources, the developer checks to see the particular licenses that the resources are offered under. The system then calculates the resources' dependencies based on the constraints current project being developed. The discovered resources and their dependencies are added to the project's definition and the developer then goes about her business using the discovered resources in the project.

4.2 Build

Resources necessary to do development are pulled from the information contained in the repository. These resources are available through the use of various build environments such as Maven and Ivy. The build may not necessarily declare the entire set of dependencies, rather these build systems will rely on the metadata in the repositories to pull in the dependent resources necessarily to build the systems. These systems will have the flexibility to define their dependencies using the full set of resource requirements, not limiting the expressiveness to jars but describing their dependencies using packages.

4.3 Provisioning

Once a system is built, many different configurations may be pulled from the set of system artifacts. Rather than deliver the entire set of system artifacts to every provisioned system, it is desirable to indicate the critical set of resources that define the system and let the resolution process determine what other resources are required. The provisioned systems can now use these sets of transitive closures to pull the required resources from common repositories.

Multiple products are often installed on the same machine and the producers of the software would like to share common resources between the separate installations. These installations are often customized by the end user and may be extended by end user content – e.g. an application server which end customer applications are deployed. These customizations and customer deployed content may change the resolved resources due to constraints such as a particular version need on particular resources by the customer. The repositories should be able to adapt to these changing requirements and integrate into repositories created by the end customer.

4.4 Patching

Successful software systems are rarely static in that they continually evolve, producing new versions and specialization. The producers of these systems would like to be able to statically determine the delta between different versions of the delivered software and deliver these subsets as patch sets to their customers. Customers should also be able to connect to vendor's repositories and calculate the resource sets required for them to move their particular installation to include a given set of critical security patches, relevant to their installations.

5 Requirements

5.1 Negative Requirements

1. **Consistent resolution throughout the development cycle**
The solution SHALL NOT be required to produce identical sets of resolved resources throughout every phase of the development process. The solution to having a consistent set of resolved resources is to resolve the system once and use this resolution set statically for the remainder of the process.

2. **Identical resolution between the runtime and repository**
The solution SHALL NOT be required to produce a set of resolved resources identical to the runtime resolution of a given OSGi container. The actual runtime resolver process has simpler constraints than the resolution process envisioned in this RFP. In addition, the runtime likely does not have the same context as was used in the resolution process for the repository in that there may be additional resources and environment constraints that change the dependency resolution and alter the resolved set.

5.2 Functional

1. The solution MUST NOT preclude provide the ability to browse the repository via a web browser

2. The solution MUST provide access to the physical bits of the resources so that they may be directly accessed after discovery

3. The solution MUST handle dependency resolution such that resources can be deployed without generating errors[1]

4. The solution MUST allow repositories to be linked, creating a federated repository that spans organizations and disparate media

5. The solution MUST NOT require federation at the repository server

6. The solution MUST provide programatic access to the repository

5.3 Discovery

1. The solution MAY allow repositories to be searched by keywords

2. The solution MAY allow repositories to be searched by categories

1 Note that deployment of resolved set of resources to a running system may fail because of existing resources already present in the runtime. These are resources that the resolver was not able to take into account when creating the transitive closure of resources and consequently may have impacts on the deployment which would break the otherwise correct deployment into that runtime.

3. The solution MUST provide filtering of resources based on execution environment

4. The solution MUST allow the discovery of licensing of resources before the artifacts are downloaded

5. The solution MUST allow multiple licensing offerings for resources

6. The solution MUST provide filtering of resources by licensing

7. The solution MUST NOT preclude the querying of resources through a RESTful API

5.4 Dependency Resolution

1. The solution MUST be able to resolve the unresolved requirements of a supplied initial set of requirements

2. Must be able to provide a list of cooperative, or optional resources

3. Cooperative, or optional, capabilities MUST be possible to select by a resource or to be offered by a provider

4. The solution MUST handle all the requirements, capabilities and their directives as defined in the OSGi R4 specifications

5. The solution MUST be able to resolve resolve a resource whose definition is not created by the solution

6. The solution MUST NOT PRECLUDE extensibility beyond the 4.2 capabilities

5.5 Non Functional

1. A repository conforming to the solution MUST NOT preclude the ability for the implementation to scale to hundreds of thousands bundles

2. The solution MUST be possible to implement a repository with a simple file or set of files – i.e, a server must not be required to implement a repository or federated set of repositories

6 Security Considerations

Security is an important part of the consideration of using the ORR, but is considered to be orthogonal to the solution. Issues such as mutual authentication between the client and the bundle repositories the client is using can be handled, for example, in much the same way that TLS is handled for secure access to web pages.

7 Document Support

7.1 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]. The OSCAR Bundle Repository, Richard Hall. http://is.gd/novc

7.2 Author's Address

Name	Hal Hildebrand
Company	Oracle Corporation
Address	500 Oracle Pkwy, M/S 2op946, Redwood City, CA 94065
Voice	650 506 2055
e-mail	hal.hildebrand@oracle.com

7.3 End of Document