A Component Model for OSGi

Deleted: Draft 0.3

Open Distribution, Draft 1.0

10 Pages

Abstract

The OSGi platform provides an attractive foundation for building enterprise applications. However it lacks a rich component model for declaring components within a bundle and for instantiating, configuring, assembling and decorating such components when a bundle is started. This RFP describes a set of core features required in an enterprise programming model and that are widely used outside of OSGi today when building enterprise (Java) applications. These features need to be provided on the OSGi platform for it to become a viable solution for the deployment of enterprise applications.

Deleted: May 30, 2007

0 Document Information

0.1 Table of Contents

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 [1].

`Source code is shown in this typeface.`

All Page Within This Box

Formatted: French (France)
Formatted: French (France)
Field Code Changed
Formatted: French (France)
Field Code Changed
Formatted: French (France)
Field Code Changed
Formatted: French (France)
Field Code Changed
Formatted: French (France)
Field Code Changed
Formatted: French (France)
Field Code Changed
Formatted: French (France)
Field Code Changed
Formatted: French (France)
Field Code Changed
Formatted: French (France)
Field Code Changed
Formatted: French (France)
Field Code Changed
Formatted: French (France)

Deleted: 0 Document Information . 2¶
0.1 Table of Contents . 2¶
0.2 Terminology and Document Conventions . 2¶
0.3 Revision History . 3¶
1 Introduction . 3¶
2 Application Domain . 4¶
2.1 Terminology + Abbreviations . 5¶
3 Problem Description . 5¶
4 Use Cases . 6¶
4.1 Instantiation of bundle components when bundle is started . 6¶
4.2 Components exported as services . 6¶
4.3 References to services obtained via dependency injection . 7¶
4.4 Enterprise library using context class loader . 7¶
4.5 Existing Enterprise Application Ported to OSGi . 7¶
5 Requirements . 7¶
6 Document Support . 9¶
6.1 References . 9¶
6.2 Author's Address . 9¶
6.3 End of Document . 10¶

Formatted: English (U.S.)

0.3 Revision History

The last named individual in this history is currently responsible for this document.

Revision	Date	Comments
Initial	Feb 14 2007	Initial draft created. Adrian Colyer, Interface21 adrian.colyer@interface21.com
Draft 0.2	Feb 22 2007	Minor updates following EEG workstream call on Feb 21 2007: Clarified enterprise library use case (4.4); changed wording in section 2 to indicate that this is the model the enterprise Java market is *moving to*; fixed a couple of typos. Adrian Colyer, Interface21 adrian.colyer@interface21.com
Draft 0.3	May 30 2007	Minor updates following EEG Face-to-Face meeting, May 14-15 Added requirement that any solution must co-exist with Declarative Services configured bundles Added requirement to define support from OSGi Minimum Execution Environment upwards
Draft 1.0	July 12 2007	Updates following EEG Face-to-Face meeting, June 2007 (Munich) Changed from "MAY" to "SHOULD" in the requirement: "…It SHOULD support re-injection of configuration value if configuration information is changed via the Configuration Admin service after the bundle components have been initially instantiated and configured." Added requirement: The solution SHOULD allow multiple component instances to be created dynamically at runtime

1 Introduction

In 2006 Interface21, the company behind the Spring Framework [3] ("Spring"), identified a complementary relationship between the application assembly and configuration features supported by Spring, and the modularity and versioning features of OSGi. Spring is primarily used to build enterprise Java applications. In this marketplace there is a need for a solution to versioning, simultaneous deployment of more than one version of a library, a better basis for dividing an application into modules, and a more flexible runtime and deployment model. OSGi

All Page Within This Box

provides a proven solution to these problems. The question became, how can enterprise application developers take advantage of OSGi (build enterprise applications as a set of OSGi bundles) when developing Spring applications?

In response to this challenge, the Spring-OSGi project [4] was born. Spring-OSGi enables the use of Spring to configure both the internals of a bundle and also references between bundles. Even with little promotion the project quickly gathered a lot of attention. As of February 2007 there are over 450 users subscribed to the project's active discussion group. Enterprise developers have responded extremely positively to the direction being taken by the project. The Spring-OSGi project is led by Interface21, with committers from Oracle and BEA also active. The design of Spring-OSGi has been influenced by discussion (both face-to-face and in the discussion group) with key personnel in the OSGi Alliance and from the equinox, Felix, and Knopflerfish OSGi implementations.

The strong interest in the Spring-OSGi project demonstrates that the enterprise Java market is attracted to the OSGi platform, and that the set of capabilities offered by Spring-OSGi represent important additions to the OSGi platform. At the OSGi Enterprise Expert Group requirements meeting held in Dublin in January 2007 a working group was formed to create an RFP for adding these capabilities to OSGi. This document is the result of that collaboration.

2 Application Domain

The primary domain addressed by this RFP is enterprise Java applications, though a solution to the requirements raised by the RFP should also prove useful in other domains. Examples of such applications include internet web applications providing contact points between the general public and a business or organization (for example, online stores, flight tracking, internet banking etc.), corporate intranet applications (customer-relationship management, inventory etc.), standalone applications (not web-based) such as processing stock feeds and financial data, and "front-office" applications (desktop trading etc.). The main focus is on server-side applications.

The enterprise Java marketplace revolves around the Java Platform, Enterprise Edition (formerly known as J2EE) [5] APIs. This includes APIs such as JMS, JPA, EJB, JTA, Java Servlets, JSF, JAX-WS and others. The central component model of JEE is Enterprise JavaBeans (EJBs). In the last few years open source frameworks have become important players in enterprise Java. The Spring Framework is the most widely used component model, and Hibernate [6] the most widely used persistence solution. The combination of Spring and Hibernate is in common use as the basic foundation for building enterprise applications. Other recent developments of note in this space include the EJB 3.0 specification [7], and the Service Component Architecture project (SCA) [8].

Some core features of the enterprise programming models the market is moving to include:

- A focus on writing business logic in "regular" Java classes that are not required to implement certain APIs or contracts in order to integrate with a container
- Dependency injection: the ability for a component to be "given" its configuration values and references to any collaborators it needs without having to look them up. This keeps the component testable in isolation and reduces environment dependencies. Dependency injection is a special case of Inversion of Control.

All Page Within This Box

- Declarative specification of enterprise services. Transaction and security requirements for example are specified in metadata (typically XML or annotations) keeping the business logic free of such concerns. This also facilitates independent testing of components and reduces environment dependencies.

- Aspects, or aspect-like functionality. The ability to specify in a single place behavior that augments the execution of one *or more* component operations.

In Spring, components are known as “beans” and the Spring container is responsible for instantiating, configuring, assembling, and decorating bean instances. The Spring container that manages beans is known as an “application context”. Spring supports all of the core features described above.

2.1 Terminology + Abbreviations

- Inversion of Control: a pattern in which a framework is in control of the flow of execution, and invokes user-code at appropriate points in the processing.

- Dependency Injection: a form of inversion of control in which a framework is responsible for providing a component instance with its configuration values and with references to any collaborators it needs (instead of the component looking these up).

- Aspect-oriented programming (AOP): a programming paradigm in which types known as “aspects” provide modular implementations of features that cut across many parts of an application. AspectJ [10] is the best known AOP implementation.

- Application Context: a Spring container that instantiates, configures, assembles and decorates component instances known as beans.

- Bean: a component in a Spring application context

- JMS: Java Messaging Service

- JPA: Java Persistence API

- JavaServlets: Java standard for serving web requests

- EJB: Enterprise JavaBeans component model defined by the Java Platform, Enterprise Edition

- JTA: Java Transaction API

- JSF: JavaServer Faces, component model for web user interfaces

- JAX-WS: Java API for XML-based web services

All Page Within This Box

3 Problem Description

Enterprise application developers working with technologies such as those described in section 2 would like to be able to take advantage of the OSGi platform. The core features of enterprise programming models previously described must be retained for enterprise applications deployed in OSGi. The current OSGi specifications are lacking in the following areas with respect to this requirement:

- There is no defined component model for the internal content of a bundle. Declarative Services only supports the declaration of components that are publicly exposed.
- The configuration (property injection) and assembly (collaborator injection) support is very basic compared to the functionality offered by frameworks such as Spring.
- There is no model for declarative specification of services that cut across several components (aspects or aspect-like functionality)
- Components that interact with the OSGi runtime frequently need to depend on OSGi APIs, meaning that unit testing outside of an OSGi runtime is problematic
- The set of types and resources visible from the context class loader is unspecified. The context class loader is heavily used in enterprise application libraries
- Better tolerance of the dynamic aspects of the OSGi platform is required. The programming model should make it easy to deal with services that may come and go, and with collections of such services, via simple abstractions such as an injecting a constant reference to an object implementing a service interface, or to a managed collection of such objects. See the description of osgi:reference in the Spring-OSGi specification [4] for an example of the level of support required here.

Providing these capabilities on the OSGi platform will facilitate the adoption of OSGi as a deployment platform for enterprise applications. This should be done in a manner that is familiar to enterprise Java developers, taking into account the unique requirements of the OSGi platform. The benefits also extend to other (non-enterprise) OSGi applications that will gain the ability to write simpler, more testable bundles backed by a strong component model.

4 Use Cases

This is not an exhaustive list of use cases that should be supported by any RFC in response to this proposal, but should instead be treated as representative of typical scenarios.

All Page Within This Box

4.1 Instantiation of bundle components when bundle is started

A securities exchange data access bundle contains data-access classes for storing and retrieving domain objects relating to securities trading. This includes the TradeDAO and TradePositionDAO classes amongst others. The bundle is installed in an OSGi runtime and then started. The bundle contains an applicationContext.xml file (for example) defining the internal components of the bundle and how they should be assembled. An extender bundle (separate to the data access bundle) or similar mechanism detects that a bundle with component metadata has been started, reads the component description information for the bundle and creates a TradeDao component instance and a TradePositionDao instance in accordance with the component metadata. Both of these are injected with a reference to a JdbcTemplate component, which is also defined in the applicationContext.xml file.

4.2 Components exported as services

A securities exchange data access bundle is installed and started as described in use case [4.1]. The applicationContext.xml file specifies that the TradeDao and TradePositionDao components should be exported to the OSGi service registry. These components are exported to the OSGi service registry automatically with no code written in the data access bundle to achieve this.

4.3 References to services obtained via dependency injection

The securities application bundle containing a TradingService component is installed and started. The component metadata for this bundle declares that the TradingService component needs access to a TradeDao. The TradeDao is looked up via the OSGi service registry, and the TradingService is injected with a reference to the discovered service. No code is written in the securities application bundle to achieve this.

The data access bundle defining owning the TradeDao service is refreshed causing the service to be unregistered and a replacement service subsequently registered. The TradingService component is able to continue working with the same TradeDao service reference it was originally provided with and invocations will now be directed to the new service.

4.4 Enterprise library using context class loader

The data access bundle imports packages from the (fictional) "Persistenator" persistence library bundle. When the components of the data access bundle are created by the extender, an instance of PersistenatorFactory is instantiated and configured by specifying a resource path to a configuration file defined within a domain model bundle. Inside the configuration file the configuration data refers to the fully-qualified names of types in the domain model bundle. The PersistenatorFactory is written to load both the configuration file and types it refers to using the context class loader. This works because the context class loader was automatically configured by the framework to be able to see the classes and resources of the domain model bundle.

4.5 Existing Enterprise Application Ported to OSGi

The NetStore enterprise application is written using a Spring and Hibernate stack. The developers of the application would like to repackage it as a set of bundles that can be deployed to an OSGi platform. NetStore is divided into four bundles – one for the presentation layer components, one for the service layer, one for the data

All Page Within This Box

access layer, and one containing the domain model. NetStore already had three Spring configuration files describing the presentation, service, and data access layers respectively. By placing each configuration file in a well-known location in the respective bundle, a Spring application context is automatically created for each bundle when it is started, containing all the required components.

5 Requirements

- The solution MUST enable the instantiation and configuration of components inside a bundle based on metadata provided by the bundle developer.
- The solution SHOULD NOT require any special bundle activator or other code to be written inside the bundle in order to have components instantiated and configured.
- The solution MAY choose to provide an extender bundle that is responsible for instantiating and configuring components inside a bundle with component metadata, when such bundles are started.
- The solution SHOULD enable the creation of components inside a bundle to be deferred until the dependencies of those components are satisfied.
- The solution MUST provide guarantees about the set of resources and types visible from the context class loader during both bundle initialization and when operations are invoked on services.
- The solution MAY provide a means for components to obtain OSGi contextual information (such as access to a BundleContext) without requiring the programmer to depend on any OSGi “lookup” APIs. This is required so that components may be unit tested outside of an OSGi runtime.
- The solution MUST provide a mechanism for a bundle component to be optionally exported as an OSGi service. It MAY provide scope management for exported service (for example, a unique service instance for each requesting bundle).
- The solution MUST provide a mechanism for injecting a reference to an OSGi service into a bundle component. It SHOULD provide a constant service reference that the receiving component can use even if the target service backing the reference is changed at run time.
- The solution MUST provide a mechanism for injecting a reference to a set of OSGi services into a bundle component. It SHOULD provide access to the matching OSGi services via a constant service reference that the receiving component can use even if the target services backing the reference change at run time.
- The solution MUST provide a mechanism for service clients obtaining references as described to be notified when a backing target service is bound or unbound.
- The solution SHOULD tolerate services in use being unregistered and support transparent rebinding to alternate services if so configured.

All Page Within This Box

Deleted: May 30, 2007

- The solution SHOULD support configuration of bundle components with configuration data sourced from the OSGi Configuration Admin service. It SHOULD support re-injection of configuration value if configuration information is changed via the Configuration Admin service after the bundle components have been initially instantiated and configured.

Deleted: MAY

- The solution SHOULD provide a rich set of instantiation, configuration, assembly, and decoration options for components, compatible with that expected by enterprise programmers used to working with containers such as Spring.
- The solution SHOULD allow multiple component instances to be created dynamically at runtime.

Formatted: Bullets and Numbering

- The solution SHOULD present a design familiar to enterprise Java developers.
- The solution MUST enable bundles configured using the component model to co-exist with bundles using Declarative Services
- The solution MUST define capabilities available on the OSGi minimum execution environment
- The solution MAY define enhanced capabilities available on other execution environments, as long as there is a strict subset/superset relationship between the features offered in less capable execution environments and the features offered in more capable execution environments.

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]. Spring Framework: http://www.springframework.org

[4]. Spring-OSGi: http://www.springframework.org/osgi

[5]. Java Platform, Enterprise Edition (Java EE)

[6]. Hibernate http://www.hibernate.org

[7]. Enterprise JavaBeans 3.0 Specification http://java.sun.com/products/ejb/

[8]. Service Component Architecture http://www.osoa.org

[9]. Aspect-Oriented Programming http://en.wikipedia.org/wiki/Aspect-oriented_programming

[10]. AspectJ http://www.eclipse.org/aspectj

All Page Within This Box

6.2 Author's Address

Name	Adrian Colyer
Company	Interface21
Address	Summit House, 2-2A Highfield Road, Dartford, Kent DA1 2JY
Voice	+44 7766 231925. skype: adriancolyer
e-mail	adrian.colyer@interface21.com

6.3 End of Document

All Page Within This Box