Dragon Architecture Guide

This document describes the Dragon SOA Governance Platform Architecture, by eBM WebSourcing.

Dragon Team
Olivier FABRE <olivier.fabre@ebmwebsourcing.com>

- May 2009 -

Table of Contents

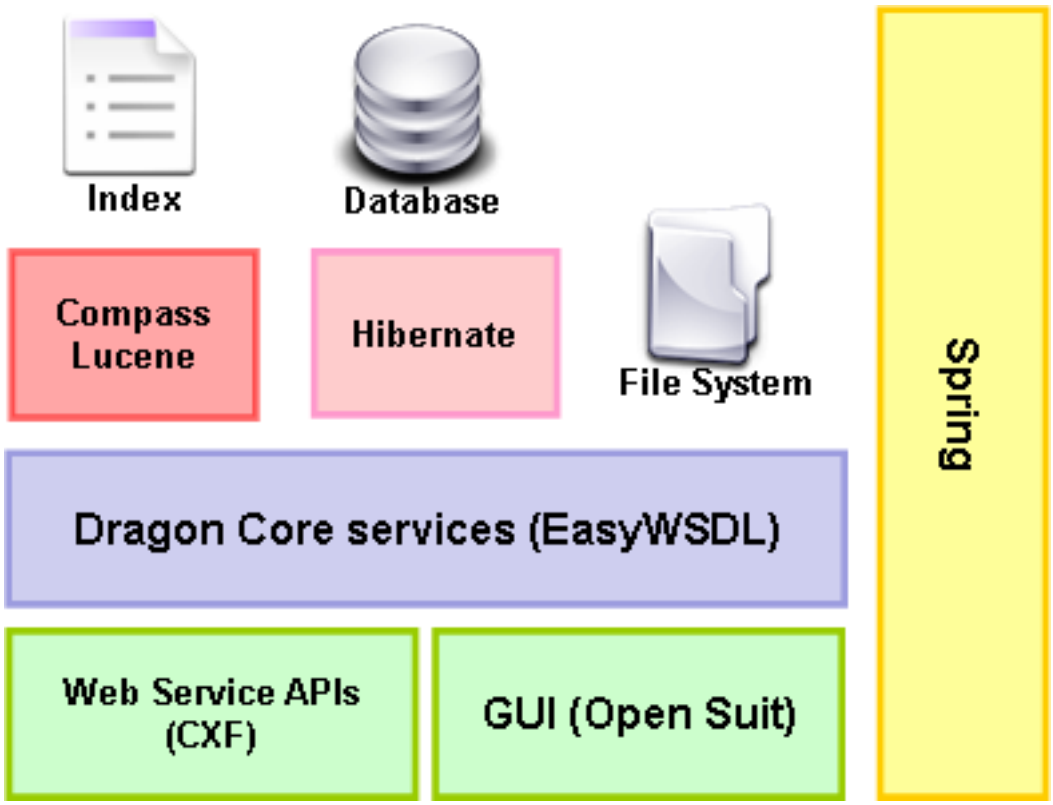
List of Figures

Structure of the document

This document describes the Dragon SOA Governance Platform Architecture. It is designed for Dragon developers.

Chapter 1. Dragon technical architecture overview

Figure 1.1. Dragon architecture overview



The whole Dragon architecture is assembled by the Spring application framework. Spring provides a lightweight container capable of assembling a complex system from a set of loosely-coupled components (POJOs) in a consistent and transparent fashion. It also provides

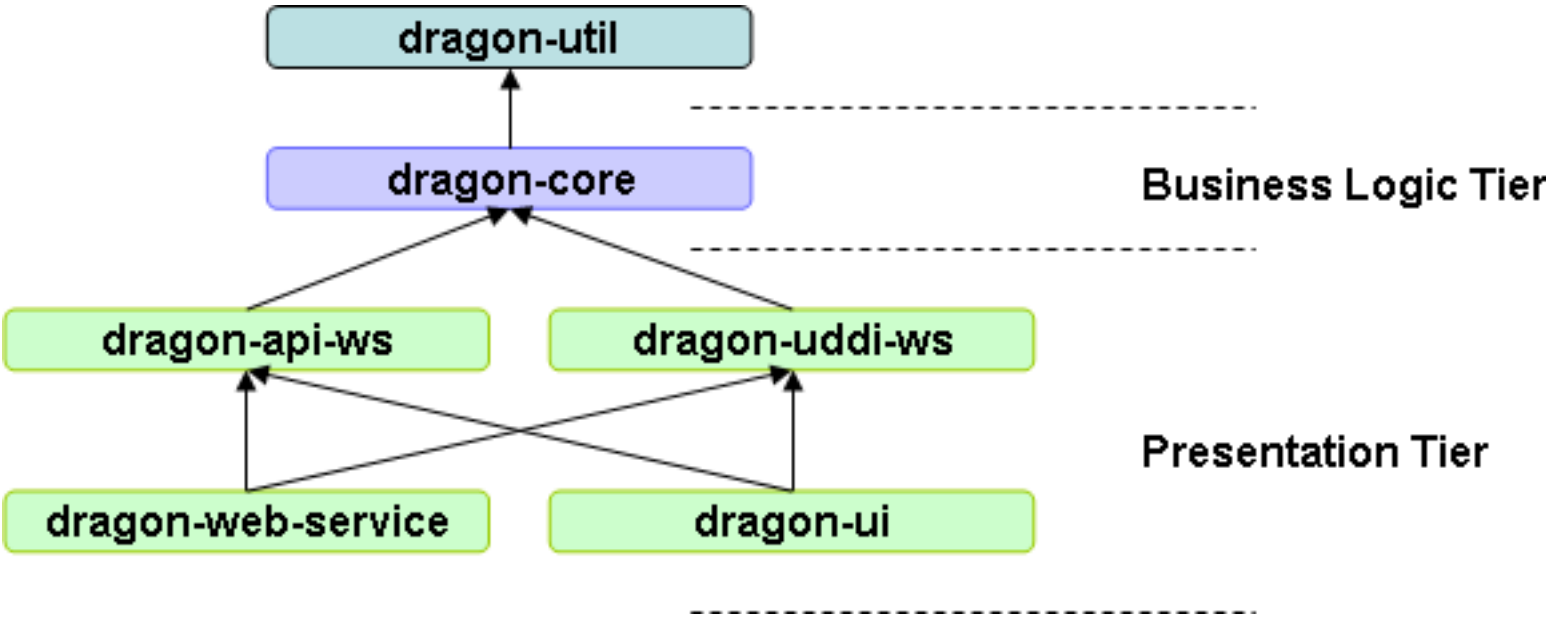
- common abstraction layer for transaction management, allowing for pluggable transaction managers, and making it easy to demarcate transactions without dealing with low-level issues,
- integration with Hibernate in terms of resource holders, DAO implementation support, and transaction strategies,
- AOP functionality, etc.

Other open source components (Hibernate, Compass/Lucene, EasyWSDL, CXF and Open Suit) that Dragon uses will be described in following chapters.

Chapter 2. Dragon Maven Modules

Dragon uses Apache Maven as a building, reporting and documentation tool. Dragon is composed of six main Maven modules : "dragon-util", "dragon-core", "dragon-api-ws", "dragon-uddi-ws", "dragon-ui", "dragon-web-service". They are described in details in this chapter.

Figure 2.1. Maven modules dependencies



Dragon is a web application based on a classical three-tier architecture. The business logic tier and the presentation tier will be described in the following sections.

2.1. The business logic tier : dragon-core

The logic tier is pulled out from the presentation tier and, as its own layer, it controls an application's functionality by performing detailed processing like exploding WSDL information into Database to allow powerful search.

This tier is represented by the "**dragon-core**" maven module. The source code is divided into four main packages :

- **org.ow2.dragon.persistence** : contains Business Objects (BO) and Data Access Objects (DAO),
- **org.ow2.dragon.api** : contains Dragon business service Java interfaces and transfer objects (TO) handled by these interfaces,
- **org.ow2.dragon.service** : contains Dragon business service Java implementations. It also contains TransferObjectAssemblers that convert BO to TO and vice versa,

  One of the business service (WSDLManager) uses EasyWSDL to handle WSDL marshalling and unmarshalling. For more information about EasyWSDL go to its website : http://easywsdl.ow2.org/.
- **org.ow2.dragon.connection.api** : contains Java interfaces and TOs that allow to connect Dragon to Service Execution Environments (like Petals ESB).

2.2. The presentation tier

This is the topmost level of the application. The presentation tier displays information related to such services as managing organization, web services, SLA and services execution environment. It communicates with other tiers by outputting results to the browser/client tier and all other tiers in the network.

Dragon proposes two types of presentation : a web user interface (Web UI) and a Web Services API (Dragon WS API + UDDI WS API). These two presentation layers are top level maven modules and packages Dragon as Web Application (war files) that could be deployed on Servlet containers (like Tomcat). The two types of presentation layer are described in the following sections.

2.2.1. Web UI : dragon-ui (+ dragon-api-ws, dragon-uddi-ws) modules

This presentation layer is represented by the "**dragon-ui**" module. It also includes WS API described in the next section

Dragon uses the OW2 presentation framework, Open Suit. For more information, go to its web site : http://opensuit.ow2.org/.

Java sources of this module are split in three principal packages:

- **org.ow2.dragon.ui.bricks** : contains Dragon specific Open Suit web components,
- **org.ow2.dragon.ui.businessdelegate** : contains business delegates that help to retrieve and call Dragon business services. For the moment, there are only Spring delegates (that call Spring Services) and Mock delegates (that call mock services). This package will be replaced by a new Open Suit feature that allow to directly call Spring service from UI Beans,
- **org.ow2.dragon.ui.uibeans** : contains UI beans. UI beans are POJOs that handle UI data. They are tied with the Web UI. They typically handle web form information. In a classical Model View Controller (MVC) presentation layer, they occupies the role of Model.

Another important parts of the **dragon-ui** module are Open Suit descriptor pages that are used by Open Suit framework to generate HTML pages that compose the Dragon UI. These descriptor pages are located in the **src/main/webapp/WEB-INF/pages** folder of the **dragon-ui** module. The **src/main/webapp/WEB-INF** folder also include other descriptor and configuration files useful to create the final Dragon war file :

- the servlets descriptor : "web.xml" file,
- the Spring framework configuration file : applicationContext-*.xml files,
- the Dragon configuration file : dragon.properties,
- the Open Suit main descriptor : DragonApp.xml,
- etc.

2.2.2. Dragon WS API : dragon-web-service (+ dragon-api-ws, dragon-uddi-ws) modules

This presentation layer is represented by the "**dragon-web-service**" module and includes "**dragon-api-ws**" and "**dragon-uddi-ws**".

Dragon provides its own web services API. For the moment, only few Dragon APIs are exposed as web services. You could add new one here. Web services are composed of Java POJOs (Plain Old Java Object) annotated with JaxWS annotations. This annotated classes are exposed as web services thanks to the Apache CXF framework. They are located in the "**dragon-api-ws**" maven module.

Dragon provides a UDDI v3 API. For the moment only the Inquiry UDDI API is available. You could implements other UDDI APIs here (Publish, etc.). As for Dragon native WS APIs, UDDI web services are composed of Java POJOs annotated with JaxWS annotations. They are located in the "**dragon-uddi-ws**" maven module.

As for Web UI layer, the web application configuration files (web.xml, etc.) are located in the **src/main/webapp/WEB-INF** folder.

Chapter 3. Business Objects

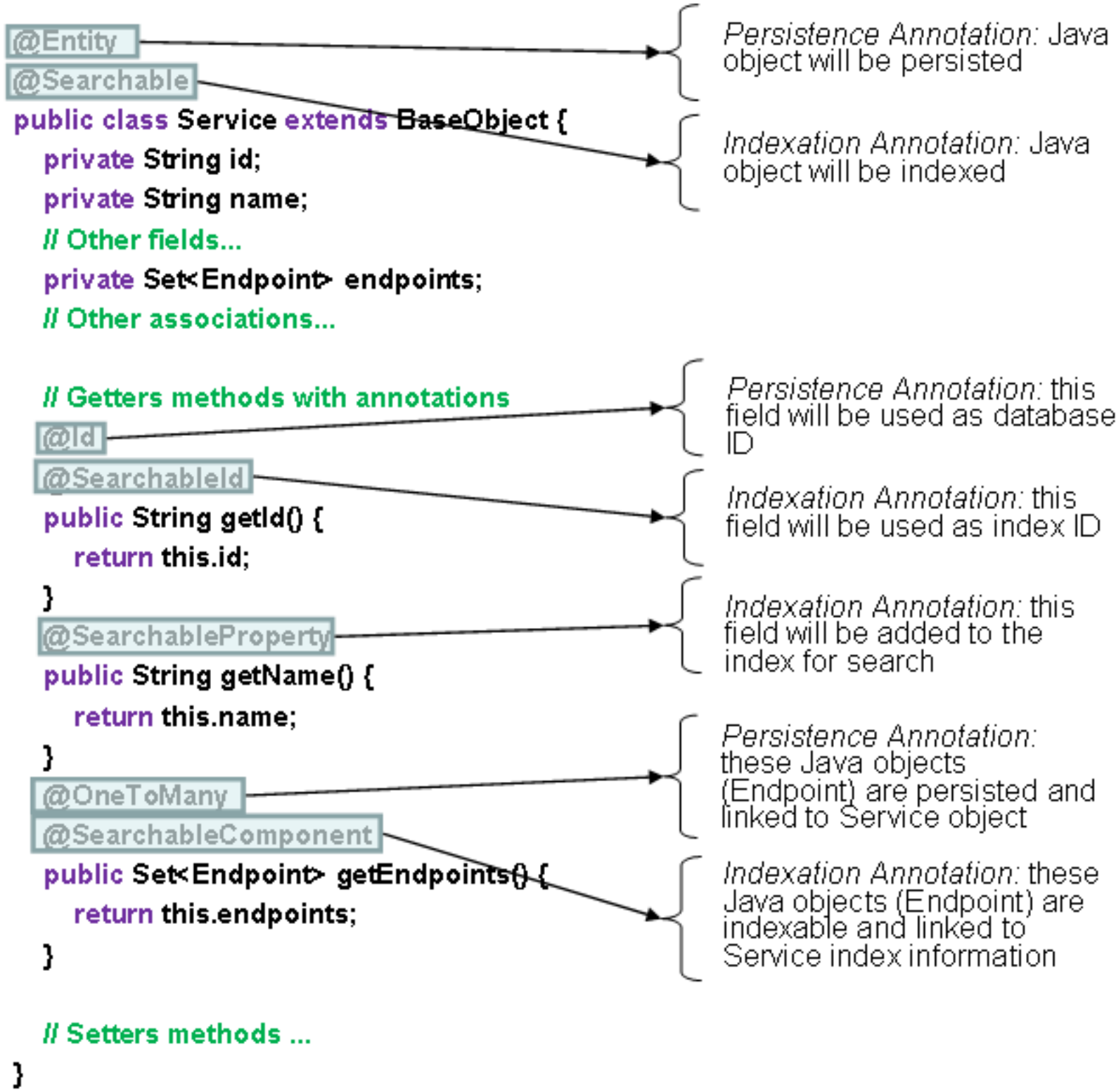
Business Objects (BO) represent data manipulated by the Dragon business logic, like Services, Endpoints, Interfaces, Organizations, Persons, etc.

BOs are POJOs annotated with Persistence annotations that describe the Object to Relational database mapping. These annotations will be used by the Object/Relational Mapping (ORM) tool, Hibernate, to create database schema and to persist Java into the database of your choice. For more information about Hibernate see its documentation here : http://docs.jboss.org/hibernate/stable/core/reference/en/html/.

They also use Compass annotations that describe Object to Search Engine mapping. These annotations will be used by Object/Search Engine Mapping (OSEM) tool, Compass, to create index structure and to persist indexing data into Ram, FileSystem or Database storage of your choice. For more information about Compass see its documentation here :http://www.compass-project.org/docs/2.2.0/reference/html/ .

Here after is a code snippet illustrating a typical BO with annotations :

Figure 3.1. Annotated BO code snippet



If you want to add your own BO you just need to write a POJO, annotate it with Persistence annotations and Compass annotations (if you need to index some of its fields) and to put it in the same package as other Dragon BO : "**org.ow2.dragon.persistence.bo**". Your custom BOs need to be in the same package (you could add subpackages in this package) as Dragon BOs but not necessarily in the same Maven module. In other words, you could create a new Maven module in which you could place your custom BOs, your own services etc. and add it as a new Dragon Maven dependency.

Dragon will automatically detect the new BO and add it to its Hibernate and Compass configuration . It is very useful to extend Dragon capabilities without having to directly modify Dragon sources.

Chapter 4. Data Access Objects

Data Access Objects (DAO) provides a mapping from application calls to the persistence layer (database). They are in charge of Creating, Retrieving, Updating, Deleting BOs : it's what we call CRUD methods.

Dragon provides a set of common DAOs that ease the creation of new DAOs. There're two types of common DAOs :

- **Universal DAOs** provide CRUD method (save, remove, get, getAll etc.) and advanced search methods. There's three types of Universal DAO that you could adapt to your needs :
  - **UniversalORMDAO** : used to manipulate ORM (Hibernate) instances of persisted objects.
  - **UniversalOSEMDAO** : used to manipulate OSEM (Compass) instances of persisted objects. The difference between ORM instances and OSEM instances is that for OSEM instances only Searchable field are filled when you retrieve object from this type of DAO.
  - **UniversalUnifiedDAO** : unifies ORM and OSEM interfaces by providing methods of the two other UniversalDAO and adding some new "searchORMResults" methods that allow to search for OSEM indexed BO but to return ORM instances corresponding to OSEM instances (same Id).

  You could use the same instance of Universal DAO for all your BOs because you have to provide the type (Class) of BO your are manipulating in all your method calls. You also have to cast objects returned by these DAOs. The three previously described DAOs are already configured as Spring component so you could directly reference them and use them in your own business services.
- **Generic DAOs** are quiet similar to Universal DAOs but uses Java Generics to allow users define what type (Class) of persisted entity will be manipulated. This avoid passing manipulated type in all DAO method calls and casting results. But in counterpart, you have to create a GenericDAO Spring config for each type of persisted BO you create. You could find GenericDAO Spring config examples in the "applicationContext-dao.xml" Spring configuration file available in the "**dragon-core**" module. Here is a code snippet illustrating a GenericUnifiedDAO Spring config for a "MyBusinessObject" BO :

```
<!-- -Generic -ORM -DAO -Spring -config --->
<bean -id="myBusinessObjectORMDAO" -class="org.ow2.dragon.persistence.dao.GenericHibernateDAOImpl">
 - -<property -name="type">
 - - -<value>
 - - - -org.ow2.dragon.persistence.bo.mypackage.MyBusinessObject
 - - -</value>
 - -</property>
 - -<property -name="sessionFactory">
 - - -<ref -bean="sessionFactory" -/>
 - -</property>
</bean>

<!-- -Generic -OSEM -DAO -Spring -config --->
<bean -id="myBusinessObjectOSEMDAO" -class="org.ow2.dragon.persistence.dao.GenericCompassDAOImpl">
 - -<property -name="compass">
 - - -<ref -bean="compass" -/>
 - -</property>
 - -<property -name="type">
 - - -<value>
 - - - -org.ow2.dragon.persistence.bo.mypackage.MyBusinessObject
 - - -</value>
 - -</property>
</bean>

<!-- -Generic -Unified -DAO -Spring -config --->
<bean -id="myBusinessObjectUnifiedDAO"
 -class="org.ow2.dragon.persistence.dao.GenericHibernateCompassDAOImpl">
 -<property -name="genericORMDAO">
 - -<ref -bean="myBusinessObjectORMDAO" -/>
 -</property>
 -<property -name="genericOSEMDAO">
 - -<ref -bean="myBusinessObjectOSEMDAO" -/>
 -</property>
```

```
</bean>
```

You could subclass all these DAO to add custom BO manipulation methods but provided methods cover most of the BO manipulation needs.