Orchestra User Guide

This document contains an installation and user guide for Orchestra 4.2.1

Orchestra Team

- December 2009 -

Table of Contents

Introduction

This documentation is targeted to Orchestra users. It presents the installation procedure and a quick user guide of Orchestra features.

Chapter 1, *General information* describes the new version Orchestra v4

Chapter 2, *Prerequisites* describes the prerequisites to the installation of Orchestra

Chapter 3, *Installation guide* describes how to install the Orchestra engine

Chapter 4, *Configuration and Services* describes main configuration features and default services

Chapter 5, *User guide* This chapter will guide you through the discovery of the functionalities of Orchestra.

Chapter 6, *Developer's guide* guides you through APIs of Orchestra.

Chapter 1. General information

1.1. Orchestra Overview

The new version of Orchestra is based on the “Process Virtual Machine” conceptual model for processes. The Process Virtual Machine defines a generic process engine enabling support for multiple process languages (such BPEL, XPDL…).

On top of that, it leads to a pluggable and embeddable design of process engines that gives modeling freedom to the business analyst. Additionally, it enables the developer to leverage process technology embedded in a Java application.

For more information about the Process Virtual Machine, check Orchestra FAQs [http://orchestra.ow2.org/xwiki/bin/view/Main/FAQ] on the Orchestra web site [http://orchestra.ow2.org] .

1.2. Features list

Orchestra is a Web Service Orchestration solution that provides BPEL 2.0 support. Business Process Execution Language (BPEL) is an XML language created by the Oasis Consortium. More information and the specifications can be found on Oasis web site [www.oasis-open.org/committees/wsbpel/]

Orchestra provides full support of the BPEL 2.0 standard.

This version provides Web Service support using the Axis 1.4 framework or CXF 2.2.5.

Orchestra is shipped with a complete test suite and a few examples.

Orchestra is persistable. This means that all the data concerning your processes definition and intances execution is stored in a Database using a persistance framework (hibernates by default). The following database systems have been successfully tested :

- H2 Database (default)
- Postgres (8.3)
- MySQL (5.0)
- Oracle (10g)

1.3. Restrictions

Orchestra comes out with an innovative architecture based on a generic and extensible engine, called "The Process Virtual Machine" and a powerful injection technology allowing services pluggability.

This new version of Orchestra is aimed at showing the power of its very innovative architecture by providing support for all the basic activities defined in the BPEL standard. As stated in the previous section, this version provides the possibility to persist the processes definition and execution. The 4.2 release provides support for the last important BPEL statement named eventHandler. Orchestra now provides full support of BPEL 2.0. The next stage will be to extend Orchestra to provide the first Open Source Business Process Server to power your SOA infrastructure. Stay tuned ! Check the roadmap [http://wiki.orchestra.objectweb.org/xwiki/bin/view/Main/Roadmap] for more information.

This version has some restrictions on the following aspects :

- Some restrictions in assign statement :
  - no extensionAssignOperation
  - validate not supported
- Some restrictions in scope statement
  - isolated not supported
  - exitOnStandardFault not supported
- The following BPEL 2.0 statements are not supported :
  - validate
  - extensionActivity
  - import
  - extensions

1.4. Tooling

For the new version, Orchestra does not ship a graphical designer. Orchestra engine has been tested with processes created using the Netbeans BPEL designer [http://www.netbeans.org/kb/55/bpel_gsg.html]. It is also possible to use the Eclipse BPEL designer [www.eclipse.org/bpel/] . Download and installation instruction are available on the project web site. However we have encountered a few bugs in the eclipse designer. So we advise the use of NetBeans. There is a work in progress to provide a Web 2.0 designer that will be accessible directly from the console. A preview is already available.

This version of Orchestra provides an new Web 2.0 administration console. This console will be improved in following releases to add monitoring capabilities.

Chapter 2. Prerequisites

2.1. Hardware

A 1GHz processor is recommended, with a minimum of 512 Mb of RAM. Windows users can avoid swap file adjustments and get improved performance by using 1Gb or more of RAM

2.2. Software

- Orchestra requires Java Development Kit (JDK) 1.5 (also called JDK 5.0) but also runs with following releases.

  The JDK software can be downloaded from http://java.sun.com/j2se/1.5.0

- Orchestra requires Apache Ant 1.7.1 or higher

  It can be downloaded from http://ant.apache.org

Chapter 3. Installation guide

Orchestra comes in two kinds of distribution:

- Tomcat distribution: Orchestra is embedded in a web application deployed in tomcat container.
- Felix OSGI distribution: Orchestra is embedded in an OSGI bundle deployed in felix OSGI platform.

As explained in Chapter 4, *Configuration and Services*, Orchestra can use different Web Service frameworks. Apache Axis1 and CXF are supported. For each web service framework, a tomcat package and a felix package are provided.

The installation and configuration steps are independent of the web service framework.

3.1. Web Service Frameworks

3.1.1. Apache Axis

Orchestra web service implementation based on Axis 1.4 offers basic web service capabilities.

3.1.2. Apache CXF

Orchestra web service implementation based on CXF offers advanced web service capabilities.

CXF implementation adds support for:

- WS-addressing
- WS-RM

3.2. Orchestra Tomcat distribution

3.2.1. Installation

Unzip the orchestra-tomcat distribution package.

```
>unzip orchestra-tomcat-4.2.1.zip
```

A new directory `orchestra-tomcat-4.2.1` will be created. It contains an ant file to install and start Orchestra.

3.2.1.1. Basic installation

Remark : Orchestra runs in Apache Tomcat servlet container. Tomcat 5.5.23 is delivered with the Orchestra Package.

To install Orchestra, go to orchestra directory and launch the install by running ant:

```
>cd orchestra-tomcat-4.2.1
>ant install
```

The install script installs Tomcat and Orchestra. The default installation activates the persistence using the H2 Database.

> **Important**
> *if your network is based on a proxy, please specify the proxy settings in your JAVA_OPTS environment property. The system properties to specify are described in the java documentation [http://java.sun.com/j2se/1.5.0/docs/guide/net/properties.html].*

3.2.1.2. Advanced installation: Using another tomcat distribution.

Orchestra is shipped with a lightweight Apache Tomcat Servlet container. This section explains how to install Orchestra in an existing tomcat distribution.

The install.properties file in the conf directory contains the information used by Orchestra installation. The default content is:

```
catalina.home=${orchestra.dir}/tomcat
catalina.base=${catalina.home}
```

To use another tomcat installation, just update the *catalina.home* and *catalina.base* properties before calling:

```
>ant install
```

3.2.1.3. Advanced installation: into JOnAS

Orchestra is shipped with a ligthweight Apache Tomcat Servlet container. This section explains how to install Orchestra in a JOnAS Application Server.

3.2.1.3.1. JOnAS 5

- Copy Orchestra conf into $JONAS_BASE/conf (copy all files from orchestra conf directory to $JONAS_BASE/conf).
- Copy jdbc jar driver into $JONAS_BASE/lib/ext
- In orchestra.properties, change orchestra.servlet.port value to your JOnAS's tomcat port configuration (cf $JONAS_BASE/conf/tomcat6-server.xml)
- Copy Orchestra war (available in orchestra lib/ directory) to $JONAS_BASE/deploy and start JOnAS

3.2.1.3.2. JOnAS 4

- Delete or upgrade xml-apis.jar and xalan-xxx.jar from $JONAS_ROOT/lib/endorsed. These libraries are needed for J2EE compatibility, but JOnAS4 can run fine without them. JOnAS4 has old versions of these libraries, which creates incompatibility issues with Orchestra.
- Copy Orchestra conf into $JONAS_BASE/conf (copy all files from orchestra conf directory to $JONAS_BASE/conf).
- Copy jdbc jar driver into $JONAS_BASE/lib/ext
- In orchestra.properties, change orchestra.servlet.port value to your JOnAS's tomcat port configuration (cf $JONAS_BASE/conf/server.xml)
- Copy Orchestra war (available in orchestra lib/ directory) to $JONAS_BASE/webapps and start JOnAS

3.2.2. Database Management

The default configuration of Orchestra uses the Database persistence service and the H2 Database. Orchestra has also been tested with Oracle, MySQL and Postgres database system. To change to mysql, postgres or Oracle, you need to put the corresponding JDBC driver in the directory `$CATALINA_BASE/common/lib` and modify the `hibernate.properties` file (see Section 4.3.4.1, “Database Access Configuration”)

3.2.3. Orchestra directory structure

Hereafter is detailled the structure of Orchestra installation. The installation directory contains the following structure :

```
    README
build.xml
install.xml
Licence.txt
tomcat/
conf/
doc/
examples/
lib/
```

Let's present those items :

- README

  This file gives the basic information related to Orchestra

- build.xml

  This file is an ant file that provides tasks to install and use Orchestra. Just typing `ant` will result giving you the usage.

- install.xml

  This file is an ant file that is called when you run `ant install` (no need to specify the use of this file)

- License.txt

  The license of Orchestra. All of Orchestra is available under the LGPL license.

- conf/

  This directory contains all the configuration files of Orchestra.

- tomcat/

  This directory is the default Tomcat installation shipped with Orchestra.

- doc/

  This directory contains the documentation of Orchestra. It contains :

  - userGuide.pdf

    For PDF documentation

- html/userGuide.html

  For HTML documentation in a single page

- html/userGuide/userGuide.html

  For HTML documentation in different pages

- examples/

  This directory contains the examples provided with Orchestra package. See Section 5.4, “Running the examples”

- lib/

  This directory contains the libraries used in Orchestra.

3.3. Orchestra OSGI Felix distribution

3.3.1. Installation

Unzip the orchestra-felix distribution package.

```
>unzip orchestra-felix-4.2.1.zip
```

A new directory `orchestra-felix-4.2.1` will be created. It contains an ant file to install and start Orchestra.

Remark : Orchestra runs in Apache Felix OGSI platform. Felix 1.8.1 is delivered with the Orchestra Package.

There is no specific installation step for running Orchestra :

```
>cd orchestra-felix-4.2.1
```

The default configuration activates the persistence using the H2 Database.

> **Important**
> *if your network is based on a proxy, please specify the proxy settings in your JAVA_OPTS environment property. The system properties to specify are described in the java documentation [http://java.sun.com/j2se/1.5.0/docs/guide/net/properties.html].*

3.3.2. Database Management

The default configuration of Orchestra uses the Database persistence service and the H2 Database. Orchestra has also been tested with Oracle, MySQL and Postgres database system. To change to mysql, postgres or Oracle, you need to install the corresponding JDBC driver bundle in the OSGI platform and modify the `hibernate.properties` file (see Section 4.3.4.1, “Database Access Configuration”)

3.3.3. Orchestra directory structure

Hereafter is detailled the structure of Orchestra installation. The installation directory contains the following structure :

```
    README
build.xml
Licence.txt
bundle/
conf/
doc/
examples/
lib/
```

Let's present those items :

- README

  This file gives the basic information related to Orchestra

- build.xml

  This file is an ant file that provides tasks to use Orchestra. Just typing `ant` will result giving you the usage.

- License.txt

  The license of Orchestra. All of Orchestra is available under the LGPL license.

- conf/

  This directory contains all the configuration files of Orchestra.

- bundle/

  This directory contains Apache Felix bundles and Orchestra OSGI bundle.

- doc/

  This directory contains the documentation of Orchestra. It contains :

  - userGuide.pdf

    For PDF documentation

  - html/userGuide.html

    For HTML documentation in a single page

  - html/userGuide/userGuide.html

    For HTML documentation in different pages

- examples/

  This directory contains the examples provided with Orchestra package. See Section 5.4, “Running the examples”

- lib/

  This directory contains the libraries used for tests.

Chapter 4. Configuration and Services

This chapter introduces the services configuration infrastructure provided by Orchestra as well as main services included in this version.

4.1. Simple configuration

The orchestra.properties file in the conf/ directory contains properties that can be easily changed. These properties are used by both orchestra client and orchestra server. Here is the default orchestra.properties file:

```
orchestra.servlet.host=localhost
orchestra.servlet.port=8080
orchestra.servlet.path=orchestra/services

orchestra.jmx.port=9999
orchestra.jmx.objectName=JMXAgent:name=orchestraRemoteDeployer
orchestra.jmx.serviceUrl=service:jmx:rmi:///jndi/rmi://localhost:9999/orchestraServer
```

- *orchestra.servlet.host* the host where orchestra server is installed.
- *orchestra.servlet.port* the port on which the web services will be exposed.
- *orchestra.servlet.path* the path on the server where the web services will be exposed. Orchestra web services will be available from http://${orchestra.servlet.host}:${orchestra.servlet.port}/$orchestra.servlet.path}/serviceName
- *orchestra.jmx.port* the port of the JMX server.
- *orchestra.jmx.serviceUrl* the JMX service url where the api mbeans will be available.
- *orchestra.jmx.objectName* the name of Orchestra mbean.

4.2. Services Container

The Process Virtual Machine technology includes a services container allowing the injection of services and objets that will be leveraged during the process definition and execution. Objects and services used by the Orchestra engine are defined through a XML file. A dedicated parser and a wiring framework are in charge of creating those objects. Service invoker, publisher, persistence and timers are examples of pluggable services.

This services container (aka IoC container) can be configured through a configuration file. A default configuration file is included in the package under the /conf directory (environment.xml).

This configuration is only used on the server side.

4.2.1. Environment.xml file

The default environment.xml file created during the installation of Orchestra is set to use the database implementation of the persistence service. This file also sets the configuration of hibernate. Here is the environment.xml file generated :

```
<environment-definition>
  <environment-factory>
    <hibernate-configuration name="hibernate-configuration:core">
      <properties resource="hibernate.properties"/>
      <mappings resource="hibernate/bpel.core.mappings.xml"/>
      <mappings resource="hibernate/bpel.monitoring.mappings.xml"/>
      <cache-configuration resource="hibernate/bpel.cache.xml" usage="read-write"/>
    </hibernate-configuration>
    <hibernate-session-factory configuration="hibernate-configuration:core"
                               name="hibernate-session-factory:core"/>
    <properties name="orchestra-properties" resource="orchestra.properties"/>
    <hibernate-configuration name="hibernate-configuration:history">
      <properties resource="hibernate-history.properties"/>
      <mappings resource="hibernate/bpel.monitoring.mappings.xml"/>
      <mapping resource="hibernate/bpel.util.hbm.xml"/>
    </hibernate-configuration>
    <job-executor auto-start="false" threads="10"/>
    <command-service>
      <orchestra-retry-interceptor retries="10"/>
      <environment-interceptor/>
      <standard-transaction-interceptor/>
    </command-service>
    <hibernate-session-factory configuration="hibernate-configuration:history"
                               name="hibernate-session-factory:history"/>
    <repository class="org.ow2.orchestra.services.impl.DbRepository"/>
    <publisher class="org.ow2.orchestra.axis.AxisPublisher"/>
    <invoker class="org.ow2.orchestra.services.impl.SOAPInvoker" name="serviceInvoker"/>
  </environment-factory>
  <environment>
    <chainer name="recorder">
      <recorder class="org.ow2.orchestra.persistence.log.LoggerRecorder"/>
      <ref object="journal"/>
    </chainer>
    <runtime-db-session name="runtime-session:core" session="hibernate-session:core"/>
    <timer-session/>
    <transaction/>
    <chainer name="undeployed-process-handler">
      <undeployed-process-handler
          class="org.ow2.orchestra.services.handlers.impl.ArchiveUndeployedProcessHandler"/>
    </chainer>
    <chainer name="finished-instance-handler">
      <finished-instance-handler
          class="org.ow2.orchestra.services.handlers.impl.DeleteFinishedInstanceHandler"/>
      <finished-instance-handler
          class="org.ow2.orchestra.services.handlers.impl.ArchiveFinishedInstanceHandler"/>
    </chainer>
    <querier-db-session name="querier-session:core" session="hibernate-session:core"/>
    <hibernate-session factory="hibernate-session-factory:core" init="eager"
                       name="hibernate-session:core"/>
    <chainer name="archiver">
      <archiver class="org.ow2.orchestra.persistence.log.LoggerArchiver"/>
      <ref object="history"/>
    </chainer>
    <message-session/>
    <job-db-session session="hibernate-session:core"/>
    <hibernate-session factory="hibernate-session-factory:history" init="eager"
                       name="hibernate-session:history"/>
    <journal class="org.ow2.orchestra.persistence.db.DbJournal" name="journal">
      <arg>
        <ref object="querier-session:core"/>
      </arg>
    </journal>
    <queryApi name="queryList">
      <ref object="journal"/>
      <ref object="history"/>
    </queryApi>
    <querier-db-session name="querier-session:history" session="hibernate-session:history"/>
    <history class="org.ow2.orchestra.persistence.db.DbHistory" name="history">
      <arg>
        <ref object="querier-session:history"/>
      </arg>
    </history>
  </environment>
</environment-definition>
```

Currently, following objects implementations can be injected in the environment:

- **publisher:** object intended for publishing services of the given bpel process. For web services based on axis framework, degfault class is org.ow2.orchestra.axis.AxisPublisher. For web services based on cxf framework, the default class is org.ow2.orchestra.cxf.CxfPublisher.
- **invoker:** object intended for external web services invocations. Default implementation is based on SAAJ through the default implementation (class org.ow2.orchestra.services.impl.SOAPInvoker). For web services based on cxf framework, the default class is org.ow2.orchestra.cxf.CxfInvoker
- **repository:** data repository storing processes and instances... Db persistence (class org.ow2.orchestra.execution.services.db.DbRepository) implementation is included in this RC.
- **recorder:** object responsible of process execution logs. Default implementation handles process logs in the command line console (org.ow2.orchestra.persistence.log.LoggerRecorder). Recorder and Journal (see next) objects can be chained (new ones can be added as well on top of the recorder chainer). This give you a powerful mechanism to handle process execution data
- **journal:** object responsible for storing or retrieving process execution data. Db persistence (class org.ow2.orchestra.persistence.db.DbJournal) implementation is provided by default.
- **archiver:** object intended for process logs archiving. Default implementation handles logs on process data archiving through the default implementation (class org.ow2.orchestra.persistence.log.LoggerArchiver). Archiver and History (see next) objects can be chained (new ones can be added as well on top of the archiver chainer). This give you a powerful mechanism to handle process archived data
- **history:** object intended for storing or retrieving process archieved data. Default implementation is provided and available in the following class: org.ow2.orchestra.persistence.db.DbHistory.
- **queryList:** object intended to configure how the QueryRuntimeAPI will retrieve the process execution data. This retrieval could be configured to chain with the expected order into the journal and the history.
- **finished-instance-handler:** action to perform when a process instance is finished. This object could chain two distinct actions: for a given process instance, deleting the runtime object including its activities from the repository and then store data in the archive and remove data from journal. Default implementations are proposed for both chained actions.

* Note 1: As explained before persistence objects are provided as default implementations in the environment. Notice that in a persistence configuration additional resources are required, i.e for hibernate persistence you can specify mapings, cache configuration...

* Note 2: The environment is divided in two different contexts: application and block. Objects declared inside the application context are created once and reused while objects declared inside the block context are created for each operation.

4.3. Services

Services in Orchestra is all about pluggability. To allow that, each service has been thought in terms of an interface with different possible implementations. In the following lines you will find a description of main services supported in Orchestra.

The PVM includes a framework to allow the injection of services and objects that will be leveraged during the process definition and execution. Objects and services required in Orchestra are defined through an XML file. A dedicated parser and wiring framework in the PVM is in charge of creating those objects.

A default environment file (environment.xml) is provided in the installed package.

Currently, following objects are required for the execution environment :

- publisher
- invoker
- repository
- persistence
- timer
- journal and history
- querier

Example of implementation classes for these objects are embedded into the Orchestra jar and defined into the environment.xml file.

4.3.1. Publisher

The publisher service sets the way the services proposed by the BPEL processes will be published. The default implementation of this service uses the Axis Web Service Container.

4.3.2. Invoker

The invoker service sets the way the BPEL processes will call external services. The default implementation of this service uses the SAAJ implementation.

4.3.3. Repository

The repository service sets the way the data will be handled by the engine. Orchestra proposes one implementation managing data in the database.

4.3.4. Persistence

Persistence is one of key technical services injected into the services container. This service, as well as other major services in Orchestra, is based on a service interface. That means that multiple persistence implementations can be plugged on top.

The Persistence service interface is responsible to save and load objects from a relational database. By default, a persistence implementation based on the Hibernate ORM framework is provided (JPA and JCR to come).

The Process Virtual Machine core definition and execution elements (processes, nodes, transitions, events, actions, variables and executions) as well as the BPEL extension ones (activities, conditions, variables...) are persisted through this service. Process Virtual Machine core elements are also cached by leveraging the default persistence service implementation (Hibernate based). Processes and instances are stored through this persistence service. Repository is the term used in Orchestra to store those entities.

This service is only used if the repository service is set to database.

4.3.4.1. Database Access Configuration

The default configuration of Orchestra uses the Database persistence service and the H2 Database. Orchestra has also been tested with Oracle, MySQL and Postgres database system. To change to mysql,

postgres or Oracle, you need to install the corresponding JDBC driver (see Chapter 3, *Installation guide*) and modify the `hibernate.properties` file : uncomment the corresponding lines :

```
# Hibernate configuration

# For using Orchestra with H2
# hibernate.dialect                      org.hibernate.dialect.H2Dialect
# hibernate.connection.driver_class      org.h2.Driver
# hibernate.connection.url               jdbc:h2:file:db_orchestra
# hibernate.connection.username          sa
# hibernate.connection.password

# For using Orchestra with postgreSQL
# hibernate.dialect                      org.hibernate.dialect.PostgreSQLDialect
# hibernate.connection.driver_class      org.postgresql.Driver
# hibernate.connection.url               jdbc:postgresql://server:port/db
# hibernate.connection.username          user
# hibernate.connection.password          pass

# For using Orchestra with MySQL
# hibernate.dialect                      org.hibernate.dialect.MySQL5InnoDBDialect
# hibernate.connection.driver_class      com.mysql.jdbc.Driver
# hibernate.connection.url               jdbc:mysql://server:port/db
# hibernate.connection.username          user
# hibernate.connection.password          pass

hibernate.dialect                      org.hibernate.dialect.H2Dialect
hibernate.connection.driver_class      org.h2.Driver
hibernate.connection.url               jdbc:h2:file:db_orchestra
hibernate.connection.username          sa
hibernate.connection.password

hibernate.hbm2ddl.auto                 update
hibernate.cache.use_second_level_cache true
hibernate.cache.provider_class         org.hibernate.cache.HashtableCacheProvider
hibernate.show_sql                     false
hibernate.format_sql                   false
hibernate.use_sql_comments             false
hibernate.bytecode.use_reflection_optimizer true
```

4.3.5. Journal and History

This module concerns the way in which the process data is stored during the process execution and archived when the execution is completed. This is indeed a crucial module in a process solution.

Orchestra unifies journal data et history data as the underlying essence of both is to handle process data. For that to be done, we created the concept of process record. A record is a minimal set of attributes describing a process entity execution. That means that each process entity related to the execution has its own associated record.

Those records are recorded during the process execution and stored depending on the persistence service implementation (db, xml...). The Orchestra API will retrieve record data from the records storage and sent them back to the users (meaning that records also acts as value objects in Orchestra APIs).

As soon as a process instance is finished, a typical scenario would be (by default) to move instance related process data from the production environment to a history one. While the physical device and the data structure could changed from one process engine deployment to another (XML, BI database...), the internal format could remain the same (records). This is exactly what is happening in Orchestra, when archiving data the engine just move execution records from the production to the history environment without data transformation in between.

Journal and history data are persisted in different database. Change hibernate.properties [hibernate-history.properties] file in conf directory to modify the journal [history] database

4.3.6. Querier

The querier is an API tool for getting process records corresponding to different criteria. It can get records from journal, from history or both. This possibility is defined in the environment. Several parameters will allow us to obtain this information by various criteria: their state (running or after delivery) or ID. Depending on the circumstances this request will return a record, a set of records or an empty table.

4.3.7. Timers

To handle activities deadlines, a timer service is required that can schedule timers to be executed in the future. Timers must have the ability to contain some contextual information and reference the program logic that needs to be executed when the timer expires.

This service, as well as any other asynchrous service in Orchestra is based on the Process Virtual Machine Job executor framework. Job executor framework is responsible for handling jobs. A job could be a timer scheduling or an asynchronous message for instance. When a job is created and stored in the database, the job executor starts a new transaction, fetch the job from the database and perform the instructions contained in the message.

The timer service is used for the BPEL statements "wait" and "onAlarm".

The Database implementation uses the Job Executor module of the Process Virtual Machine for the management of the timers. Its definition in the environment is the following : <timer-session />

For the Job Executor, the administrator can set the number of thread that will manage the jobs. This information is also defined in the environment file with the following line :

```
<job-executor threads='1' auto-start='false' />
```

The default number of thread for the job executoris 1. It is advised to leave this value to avoid concurrency problems.

4.3.8. Finished instance handler (FIH)

FIH are executed after the instance finished. Orchestra provides following implementations in the package org.ow2.orchestra.services.handlers.impl :

- NoOpFinishedInstanceHandler : do nothing
- CleanJournalFinishedInstanceHandler : remove instance data from journal
- ArchiveFinishedInstanceHandler : remove instance data from journal and put it in history
- DeleteFinishedInstanceHandler : delete instance data from orchestra repository

Chapter 5. User guide

5.1. Start and Stop Orchestra

Orchestra is a webapp that can be deployed on Tomcat. So starting Orchestra in fact starts Tomcat with the correct environment. This can be performed from the installation directory with the following command line :

```
>cd orchestra-tomcat-4.2.1
 >ant start
```

Starting Orchestra will not be done in background. This means that the console starting Orchestra will be dedicated to the traces from Orchestra. To perform further actions, new consoles need to be opened.

To stop Orchestra, type the following command line :

```
>cd orchestra-tomcat-4.2.1
  >ant stop
```

5.2. Deploying / undeploying a process

Once Orchestra is started, it is then possible to deploy a new process on the engine :

```
>ant deploy -Dbpel=<process>.bpel -Dwsdl=<process>.wsdl -Dextwsdl=<wsdl1,wsdl2>
```

Orchestra also provides the possibility to use an archive to deploy a process. This archive should be a zip file with the extension .bar. Here is the command line to deploy such an archive :

```
>ant deploy -Dbar=<process>.bar
```

Warning : The archive should be a zip file structured as described bellow :

```
/<process>.bpel
/<process>.wsdl
/<files>.wsdl
```

To undeploy a process, use the following command line :

```
>ant undeploy -Dprocess=<process_name>
```

Warning : the process name should be fully qualified. This means that it needs to contain to namespace. For instance :

```
{http://orchestra.ow2.org/weather}weather
```

5.3. Other commands

Orchestra provides a set of other commands that can be usefull

- A command to check the status of Orchestra. This command tells if the engine is started and if so, gives the names of processes deployed on the engine :

  ```
  >ant status
  ```

- A command to simulate a Web Service call. This command will simulate a WS call to interact with a deployed process :

  ```
  >ant call -Dendpoint=<service_url> -Daction=<SOAP_action> -Dmessage=<message>
  ```

For example :

```
>ant call -Dendpoint=http://localhost:8080/orchestra
          -Daction=http://orchestra.ow2.org/weatherArtifacts/process/weatherPT
          -Dmessage="<weatherRequest xmlns='http://orchestra.ow2.org/weather'>
                    <input>Grenoble,France</input>
                    </weatherRequest>"
```

5.4. Running the examples

The Orchestra package contains examples of BPEL processes:

- loanApproval: invokes two local web services. This example is taken from the BPEL 2.0 standard.

  This is an example from the BPEL 2.0 standards

- weather: invokes a remote Web Service and returns the current weather.

  This example shows how to call a real world Web Service.

- echo

  This example shows a basic synchronous bpel process.

- orderingService

  This example shows how to use pick instruction and correlations.

- producerConsumer

  This example shows how executions can be saved and restarted after a crash.

A build.xml file is provided for each of those samples. Those ant scripts provide the same targets to deploy, launch and undeploy the sample. Go to the desired example and use the command lines :

```
>ant deploy
>ant launch
>ant undeploy
```

5.5. Running the tests

Orchestra is delivered with a test suite to check if your installation is correct. There are 3 differents tests available :

- *Core test suite*. This suite tests the core functionnalities of the engine (e.g. BPEL activities, variables, etc...). To run this test suite, the server should not be started.This test suite can be launched with the following command :

  ```
  >ant test
  ```

- *Remote test suite*. This suite gives the possibility to test the Web Service stack deploying and launching real processes. This test suite can be launched with the following command (server should be started) :

  ```
  >ant test-remote
  ```

- *Stress test suite*. This suite will launch a small stress test. This test suite can be launched with the following command line :

  ```
  >ant test-stress
  ```

A command is also provided to launch those 3 test suites at once :

```
>ant test-all
```

The results of the tests are available under the directory `testresults`.

5.6. Configuring Logger

It is possible to activate the logs. To do so, the file `logging.properties` under the directory `conf/` can be edited. Here is the content of that file :

```
handlers= java.util.logging.ConsoleHandler
.level= SEVERE
java.util.logging.ConsoleHandler.level = FINEST
java.util.logging.ConsoleHandler.formatter = org.ow2.orchestra.util.JbpmFormatter

# For example, set the com.xyz.foo logger to only log SEVERE messages:
# com.xyz.foo.level = SEVERE

#org.ow2.orchestra.pvm.level=INFO
#org.ow2.orchestra.level=FINEST
#org.ow2.orchestra.pvm.Execution.level=FINEST
#org.ow2.orchestra.wire.level=FINEST
```

Uncomment the last lines to activate the logs.

Chapter 6. Developer's guide

This chapter describes how to start playing with Orchestra:

- How to develop a simple application by leveraging Orchestra APIs

6.1. Orchestra APIs

6.1.1. Getting started with Orchestra APIs

Actually, four different APIs are available in Orchestra :

- QueryRuntimeAPI that gives runtime information about instances of process and activities.
- QueryDefinitionAPI that gives information of process definition.
- ManagementAPI that gives the possibility to manage Orchestra (deploy / undeploy processes, etc...)
- InstanceManagementAPI that gives the possibility to manage instances of process (suspend / resume / exit process instance)

You can find detailled information about APIs in the javadocs. APIs are included in a Maven module. To include this module you have to add following Maven dependency :

```
<dependency>
  <groupId>org.ow2.orchestra</groupId>
  <artifactId>orchestra-api</artifactId>
  <version>4.2.1</version>
</dependency>
```

If you do not want / can't use Maven, you can create a Maven module which depends only on this dependency and create an assembly. Then copy created jar to your project.

- **QueryRuntimeAPI**: to get recorded/runtime informations for instances and activities. It allows also to get activities per state. Then operations in this API applies to process instances and activity instances.

Hereafter you will find an example on how to access to the QueryRuntimeAPI from your client application:

```
org.ow2.orchestra.facade.QuerierRuntimeAPI querierRuntimeAPI =
               org.ow2.orchestra.facade.AccessorUtil.getQueryRuntimeAPI(String, String);
```

The method getQueryRuntimeAPI takes two arguments of type String. The first argument is the URL of the JMX service. The second is the name of the object JMX. In case we use the default configuration, two constants can be used which are: AccessorUtil.SERVICE_URL and AccessorUtil.OBJECT_NAME.

For a detailed insight on Orchestra APIs, please take a look to the Orchestra javadoc APIs (available under / javadoc directory)

Similar methods exists to access to the QueryDefinitionAPI, ManagementAPI and InstanceManagementAPI.

6.2. Orchestra Client jar

If you want to call Orchestra APIs from a remote application you can use the Orchestra client jar. It contains all the needed classes for you to build you application. Just download the jar and include it in your classpath.