



envision
environmental services infrastructure with ontologies

Deliverable D6.2:

Adaptive Execution Infrastructure Version 1.0 – User Guide

Date:	Monday, 20. December 2010
Author(s):	A. Tsalgaidou, G. Athanasopoulos, I. Pogkas, P. Kouki (NKUA)
Dissemination level:	PU
WP:	WP6 - Adaptive execution infrastructure
Version:	1.0
Keywords:	Execution Infrastructure, User Guide
Description:	Envision Adaptive Execution Infrastructure User Guide



ICT for Environmental Services and
Climate Change Adaption

Small or Medium-scale Focused Research Project
ENVISION (Environmental Services Infrastructure with Ontologies)
Project No: 249120
Project Runtime: 01/2010 – 12/2012

Document metadata

Quality assurors and contributors

Quality assesor(s)	François Tertre (BRGM), Iker Larizgoitia (UIBK)
Contributor(s)	

Version history

Version	Date	Description
0.1	5/10/2010	Initial TOC proposal.
0.2	4/11/2010	Preliminary Input.
0.3	23/11/2010	Draft version with input in most sections.
0.4	30/11/2010	1 st completed internal version.
0.6	08/12/10	Version ready for internal review.
0.7	16/12/10	Version updated according to internal review comments.
0.9	20/12/10	Version approved by the technical coordinator.
1.0	22/12/2010	Version approved by the project coordinator.

Executive Summary

A core component of Envision is the Execution Infrastructure, which according to the DoW [1] is responsible for supporting the execution of environmental service chains, their data-driven adaptation and the semantic-based mediation among the constituent services. These features constitute the operational core of Envision and will support the functionality of the Envision Portal.

The goal of this deliverable is to

- provide an overview of the features of the Envision Execution Infrastructure implemented in version 1.0 and
- provide a user-guide with details on the installation and use of the Infrastructure.

This document is based upon the deliverable D6.1 ENVISION Adaptive Execution Infrastructure – Architecture Specification [2], which provides details on the architecture and properties of each of the Execution Infrastructure components. Therefore, interested readers should refer to deliverable D6.1 for additional technical details (see [2]).

To facilitate its objectives, this document provides a short introduction to the Envision Execution Infrastructure and its features, which have been implemented in version 1.0. Next, we provide details on the installation process that should be followed and the set of users (identified in [3]) that are supported by this version of the infrastructure. We also provide descriptions of the steps that should be followed for the deployment and execution of an environmental service chain. Finally, we conclude with a short summary of the presented material and some preliminary remarks about the version 1.0 of the infrastructure.

Table of Contents

1	INTRODUCTION.....	6
2	EXECUTION INFRASTRUCTURE FEATURES.....	6
3	INSTALLATION INSTRUCTIONS.....	8
3.1	SERVICE ORCHESTRATION ENGINE.....	8
3.1.1	Download and Install the J2SE Runtime Environment (JRE).....	8
3.1.2	Download and install Apache Tomcat.....	9
3.1.3	Download and install Apache ODE engine.....	10
3.1.4	Deploy ODE examples.....	11
3.1.5	Deployment of ODEDeploymentService.....	15
3.2	PROCESS OPTIMIZER.....	15
3.2.1	BPEL Compiler.....	16
3.2.2	WSML-Lite Parser.....	17
3.2.3	WSDL Parser.....	17
3.3	SEMANTIC CONTEXT SPACE ENGINE.....	17
4	SUPPORTED USERS.....	19
5	USING THE INFRASTRUCTURE.....	20
5.1	PROCESS DEPLOYMENT.....	20
5.2	PROCESS INVOCATION.....	21
6	SUMMARY.....	21
7	REFERENCES.....	23

Table of Figures

FIGURE 1: EXECUTION INFRASTRUCTURE HIGH LEVEL ARCHITECTURE 7
FIGURE 2: TOMCAT SUCCESSFUL INSTALLATION IN WINDOWS..... 9
FIGURE 3: TOMCAT SUCCESSFUL INSTALLATION IN UBUNTU10
FIGURE 4: SUCCESSFUL ODE INSTALLATION11
FIGURE 5: ODE ENGINE HOME AFTER THE EXAMPLES DEPLOYMENT.....12
FIGURE 6: ODE ENGINE DEPLOYMENT VIEW13
FIGURE 7 : PROCESS OPTIMIZER ARCHITECTURE16
FIGURE 8 : SCS ENGINE ARCHITECTURE18

1 Introduction

Supporting the execution of environmental service chains, their data-driven adaptation and semantic based mediation is the core goal of the Envision Execution Infrastructure. The starting point for the implementation of the first official release (version 1.0) of the Execution Infrastructure was the descriptions and specifications of the components and modules comprising the infrastructure, detailed in deliverable D6.1 [2].

The prime principles which steer the implementation of the Execution Infrastructure are:

- Gradually implement the infrastructure starting from the core features/properties.
- Append new features/properties when these become available, in an as much non-obtrusive as possible manner, which will ensure the backward compatibility of already designed processes.

It becomes apparent that the components which should be implemented in the first release of the infrastructure are the ones supporting the deployment and execution of environmental service chains. Additional features, such as the data-driven adaptation and semantic based mediation will follow in next releases.

In accordance to our development decisions, the first step towards the implementation of version 1.0 of the infrastructure was the selection of an appropriate Service Orchestration Engine. Among the existing engines (see [2] for more details) the one which was used in this release is Apache ODE version 1.3.4. Moreover, a dedicated deployment service has been developed so as to further enhance the deployment of service chains, as well as to support the integration of the deployment process with the rest of the infrastructure components.

Although full implementations and fully integrate-able versions of the rest of the Execution Infrastructure components have not yet been released, this does not imply that implementations on these components have not started. For each of these components, namely Process Optimizer and Semantic Context Space Engine, required libraries and middleware have been selected and implementations have commenced.

The goal of the deliverable is to briefly introduce the features that have been implemented in version 1.0 of the Execution Infrastructure and to provide a user guide that will assist designers and end-users in deploying and executing service chains. The rest of this document is structured so as to facilitate this goal. Specifically, in the following (section 2) we provide a short description of the components and features implemented in the first release, as well as (section 3) a set of guidelines on the installation of the provided components, i.e. both for development and execution purposes. Next, we present the set of users associated to the Execution Infrastructure (section 4) and the steps that should be performed for the deployment and execution of an environmental service chains (section 5). We conclude this deliverable with a brief summary of the implemented features and a set of preliminary remarks.

2 Execution Infrastructure Features

Supporting the deployment and execution of environmental service chains has been the prime goal of version 1.0 of the Execution Infrastructure (see Figure 1). Therefore, the respective list of components includes the ones that are mainly related to these two aspects. Specifically the status of each of the components of the Execution Infrastructure, as these have been listed in D6.1 [2] is as follows:

- **Service Orchestration Engine:** A first version of the Execution Engine based on the APACHE ODE engine ver 1.3.4, along with an appropriate Deployment Service is

envision

released. This version adheres to a centralized architectural style, which can comprise autonomous, un-related instances of the engine. The Peer-to-Peer infrastructure for the distribution of the execution engine is under development, but not integrated with the Orchestration Engine.

- **Semantic Context Space Engine:** A preliminary version of the SCS Engine supporting the semantic annotation of collected information using WordNet as the underpinning ontology, along with a Java-based interface has been released. Extensions catering for the annotation of collected information using RDFS and WSMO-based ontologies are currently under development. Additional extensions catering for the spatiotemporal annotation of the collected information will be provided in following releases. The currently available implementation of the SCS Engine has not been integrated with the rest of the infrastructure components.
- **Process Optimizer:** Preliminary versions of this component include the implementation of several internal artefacts catering for the transformation of service and process specification to internal formats required by the integrated planning algorithms. The currently available implementation has not been integrated with the rest of the infrastructure components.
- **Data Mediation Engine:** In version 1.0 of the Execution Infrastructure, data transformation among the services of an environmental service chain will rely upon the use of XSL transformations executed by the XSLT engine. Support on semantic-based mediation will not be available at version 1.0 of the infrastructure. This component is underdevelopment and will become available at following releases of the infrastructure.

The integration of the Execution Infrastructure with the Scenario Web sites in version 1.0 of the infrastructure will be performed via the use of SOAP based interfaces. Therefore, all processes deployed on the Execution Infrastructure will provide a SOAP based interface (similar to W3C compliant services) via which clients will be able to invoke them. Next releases of infrastructure will provide OGC compliant interfaces to external clients.

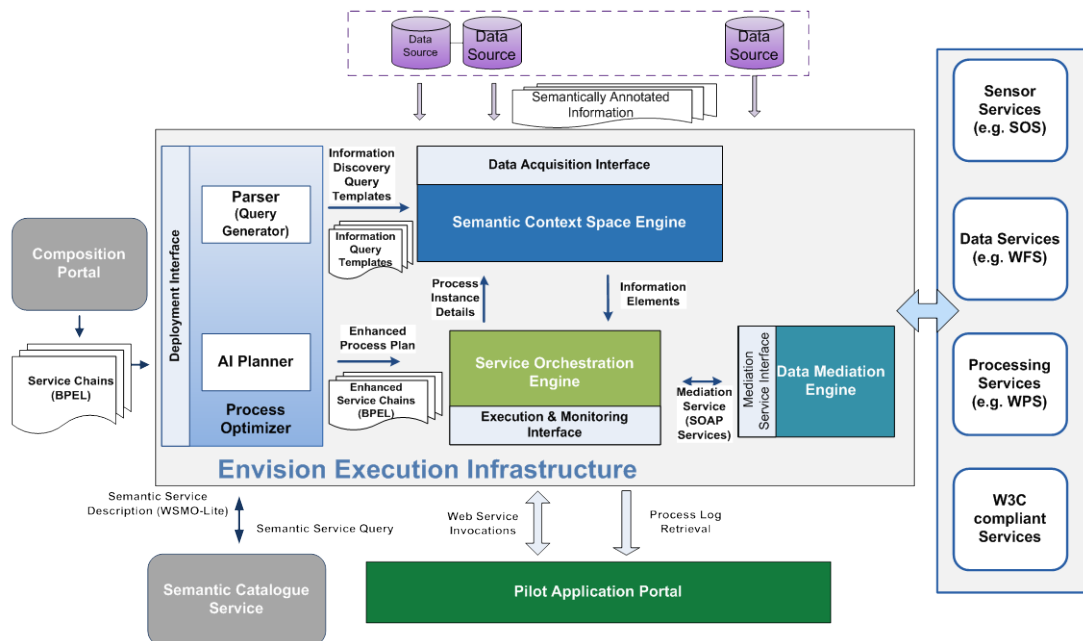


Figure 1: Execution Infrastructure high level architecture

In the following we provide details on the installation of Service Execution Engine and dependencies of the Process Optimizer and the Semantic Context Space Engine with existing open source middleware and tools.

3 Installation Instructions

To facilitate the installation process of the Execution Infrastructure we provide a set of guidelines, which address both Windows and Linux based platforms. With respect to the tools that are not included in this first release we provide details on their dependencies with existing libraries and middleware that are used for their implementation.

3.1 Service Orchestration Engine

This section presents the required steps for the installation of the Java runtime environment, the Apache Tomcat and the Apache ODE orchestration engine. As our engine requires a version of the Apache ODE to be up and running we present the correct configuration for both ODE and Tomcat. Furthermore we provide instructions for the deployment of the ODEDeploymentService on the Apache Tomcat. This service is used to automatically deploy process instances on the running ODE engine.

3.1.1 Download and Install the J2SE Runtime Environment (JRE)

Microsoft Windows (XP, Vista, 7)

1. Download the Java 2 Standard Edition Runtime Environment (JRE), release version 5.0 or later [4].
2. Install the JRE according to the instructions included with the release.
3. Set an environment variable named **JRE_HOME** to the pathname of the directory into which you installed the JRE, e.g. `c:\jre5.0` or `/usr/local/java/jre5.0`

GNU/Linux (Ubuntu 10.04)

Because Sun Java has been excluded from the default Ubuntu repository, we first need to include the partner repository as a software source and then install the sun-java packages. So we run:

```
>> sudo apt-get-repository "deb http://archive.canonical.com/ lucid partner"  
>>sudo apt-get-update
```

Then install the Java 2 Standard Edition Runtime Environment (JRE), release version 5.0 or 6.0

```
>> sudo apt-get install sun-java6-bin
```

Alternatively, you may install the Java 6 SDK

```
>> sudo apt-get install sun-java6-jdk
```

NOTE: As stated, you may also use the full JDK rather than just the JRE. In this case set your **JAVA_HOME** environment variable to the pathname of the directory into which you installed the JDK, e.g. `c:\j2sdk5.0` or `/usr/local/java/j2sdk5.0`

3.1.2 Download and install Apache Tomcat

The general installation options are described in [4]. In the following we present the installation process for specific versions of the Microsoft Windows and Ubuntu operating systems respectively:

Microsoft Windows (XP, Vista, 7)

1. Download Tomcat 6.0 distribution [5]

There are two options:

- 1.1. Install the binary distribution for the windows environment. For this purpose download one of the available Tomcat binary distributions for the windows environment.

- 1.1.1. Unpack the binary distribution into a convenient location so that the distribution resides in its own directory (conventionally named "apache-tomcat-[version]"). Set the **CATALINA_HOME** environment variable to the pathname of the directory into which you unpack the binary distribution. *For the purposes of the remainder of this document, the symbolic name "\$CATALINA_HOME" is used to refer to the full pathname of the release directory.*

- 1.2. Install the Windows service installer. This is a Windows specific distribution that includes the Windows service wrapper and the compiled APR/native library for use with 32-bit/64-bit JVMs on both 32/64 bit Windows platforms. This way tomcat is used as a Windows service.

Note: Due to a bug in the installer the apache tomcat icons on the start menu fail to start/stop the service. You can start/stop and configure Tomcat using the provided menu in the Windows toolbar. Because many users may find it confusing we recommend you to install the binary distribution as described in 1.1.

2. Start up Tomcat executing the command

```
>> $CATALINA_HOME\bin\startup.bat
```

After start up, the default web applications included with Tomcat will be available at the following URL: <http://localhost:8080/> as shown in Figure 2.

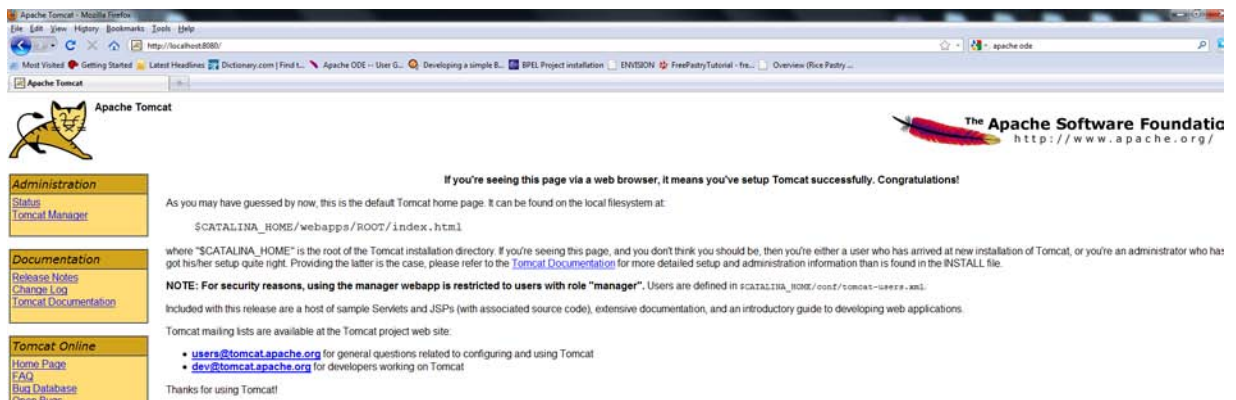


Figure 2: Tomcat successful installation in Windows

3. Shut down Tomcat using the command

```
>> $CATALINA_HOME/bin/shutdown
```

GNU/Linux (Ubuntu 10.04)

1. Install Tomcat 6.0

```
>> sudo apt-get install tomcat6
```

The default installation will be in `/var/lib/tomcat6/`. For more details read the Ubuntu provided guide in [6]. Set `CATALINA_HOME` environment variable to point to the base path of the Tomcat installation location

2. Start up Tomcat executing the command

```
>> $CATALINA_HOME/bin/startup.sh
```

After start up, the default web applications included with Tomcat will be available at: <http://localhost:8080/> as shown in Figure 3.

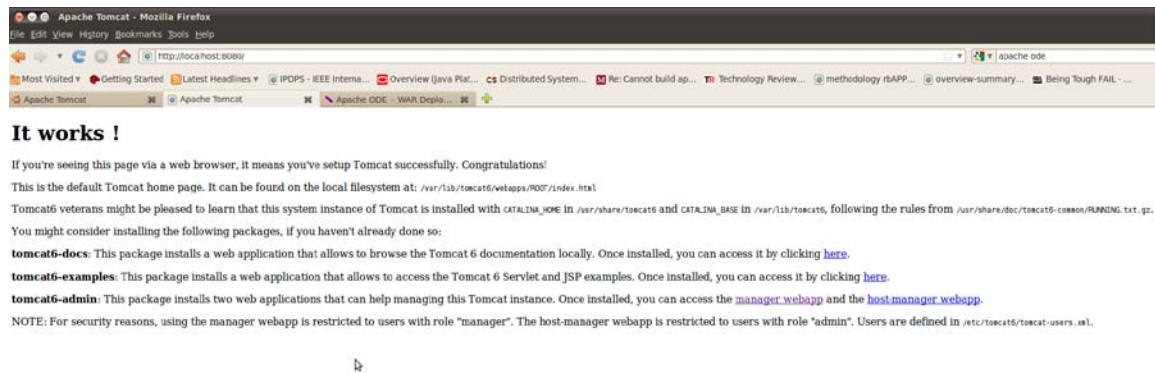


Figure 3: Tomcat successful installation in Ubuntu

3. Shut down Tomcat using the command

```
>> $CATALINA_HOME/bin/shutdown.sh
```

3.1.3 Download and install Apache ODE engine

Microsoft Windows (XP, Vista, 7) and GNU/Linux (Ubuntu 10.04)

1. Download Apache ODE WAR distribution (apache-ode-war-1.3.4.zip) from the Apache ODE download site [7]
2. Unzip the distribution somewhere on your disk and copy `ode.war` to Tomcat's `webapp` directory.
3. Start up Tomcat and ODE should be up and running. You should get the Axis2 welcome page under <http://localhost:8080/ode>. As shown by Figure 4 below.

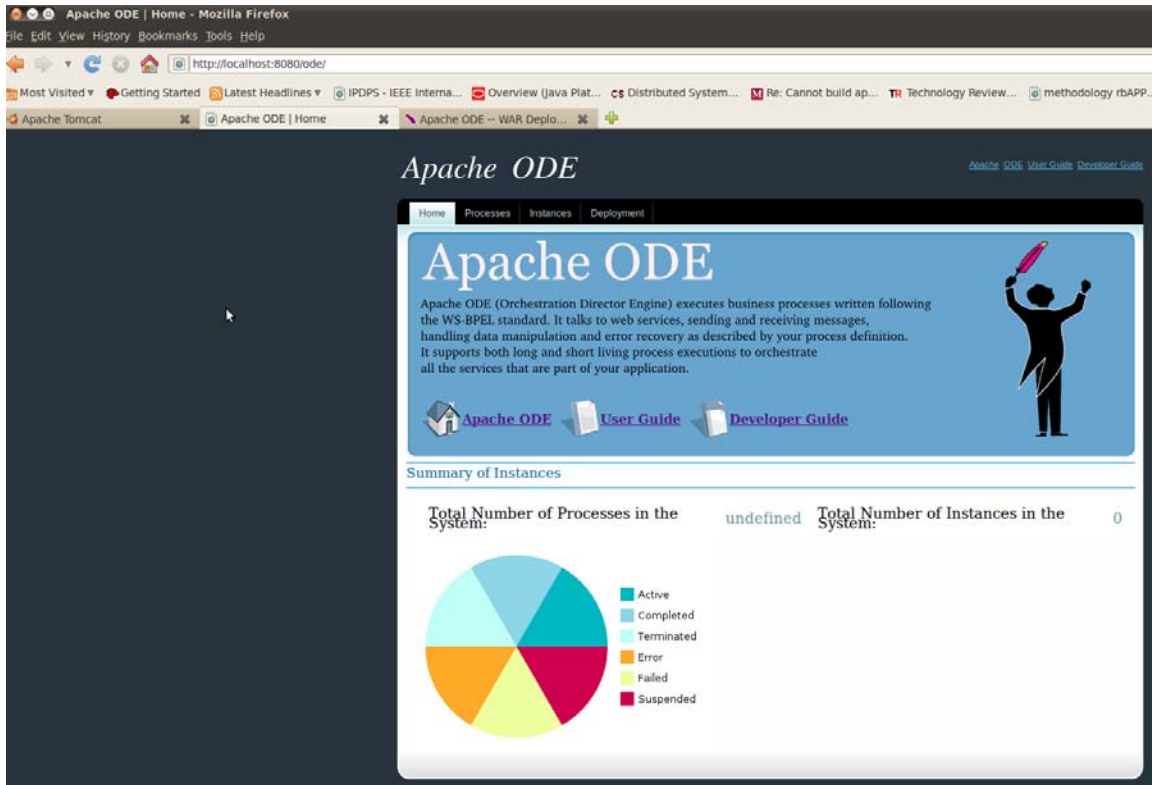


Figure 4: Successful ODE installation

Note: ODE WAR includes its own embedded database (Derby) so you don't have to worry about configuring any external database.

3.1.4 Deploy ODE examples


In order to check that the ODE is working we need to run the examples provided with the ODE installation. Copy the content of the examples directory in the ODE distribution (the 3 sub-directories: MagicSession, HelloWorld2 and DynPartner) to tomcat/webapps/ode/WEB-INF/processes. This will automatically deploy three example processes as shown by Figure 5 and Figure 6.

Apache ODE [Apache ODE User Guide](#) [Developer Guide](#)

Home Processes Instances Deployment

Apache ODE


Apache ODE (Orchestration Director Engine) executes business processes written following the WS-BPEL standard. It talks to web services, sending and receiving messages, handling data manipulation and error recovery as described by your process definition. It supports both long and short living process executions to orchestrate all the services that are part of your application.

 [Apache ODE](#)  [User Guide](#)  [Developer Guide](#)



Summary of Instances

Total Number of Processes in the System: **5** Total Number of Instances in the System: **0**



- Active
- Completed
- Terminated
- Error
- Failed
- Suspended

Figure 5: ODE engine home after the examples deployment

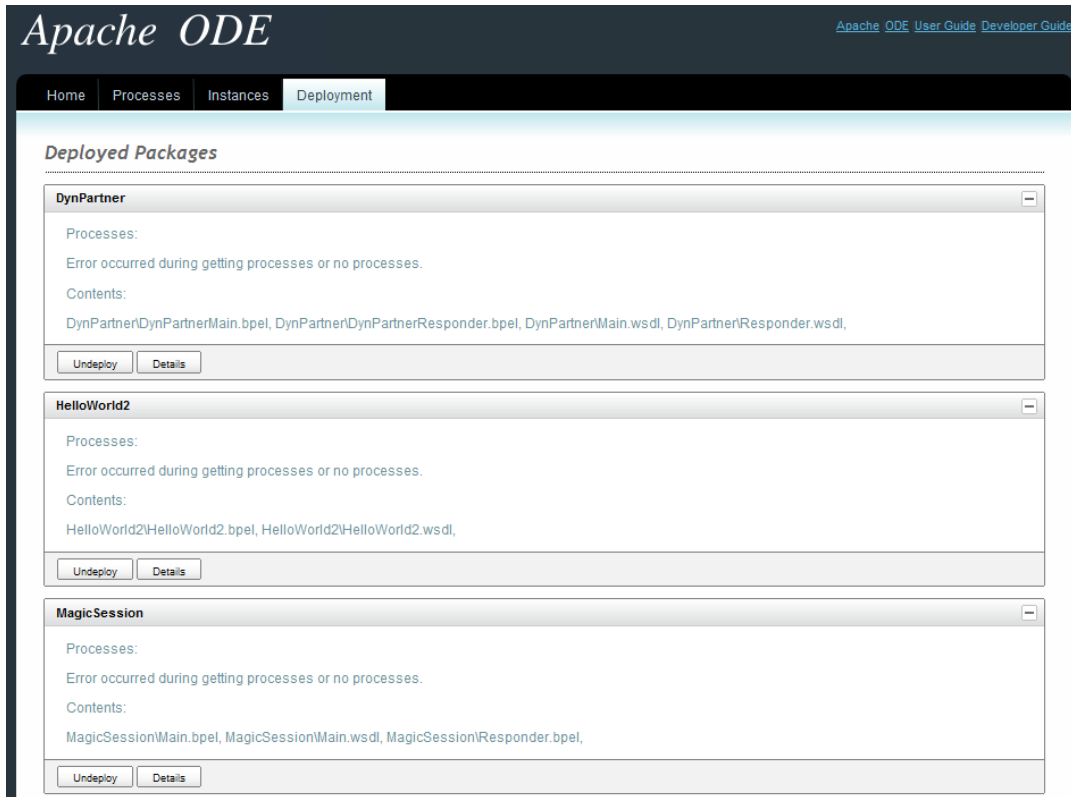


Figure 6: ODE engine deployment view

NOTE: Use the *sendsoap* command located in the distribution bin directory to send test messages. The messages to run each of the three examples are provided in their respective directory (testRequest.soap).

In the following we will test the deployed examples using the sendsoap command. For this purpose we assume that the current directory is the examples directory contained into apache-ode-war-1.3.4.zip.

HelloWorld 2

This example demonstrates a simple service invocation that synchronously replies to a message. It is built using WS-BPEL 2.0 syntax. After deployment, invoke the process with the command:

- Microsoft Windows (XP, Vista, 7)

```
>> ..\bin\sendsoap.bat http://localhost:8080/ode/processes/helloWorld
./HelloWorld2\testRequest.soap
```

- GNU/Linux (Ubuntu 10.04)

```
>> ../bin/sendsoap http://localhost:8080/ode/processes/helloWorld
./HelloWorld2/testRequest.soap
```

Please make sure that you execute the command from the example directory. The response should be a SOAP message containing the 'hello world' string.

DynPartner

This example demonstrates dynamic partner assignment. The main process asks for the responder process endpoint. The responder process gives its endpoint by assigning it to a message (assign my role) and replying this message to the main process. The main process invokes again the responder process but this time using the endpoint it just receives instead of the default one. After deployment, start the process with either of the following commands:

- Microsoft Windows (XP, Vista, 7)

```
>>..bin\sendsoap.bat http://localhost:8080/ode/processes/DynMainService
.DynPartner\testRequest.soap
```

- GNU/Linux (Ubuntu 10.04)

```
>>./bin/sendsoap http://localhost:8080/ode/processes/DynMainService
./DynPartner ./testRequest.soap
```

Please make sure that you execute the command from the example directory. The response should be an 'OK' SOAP message, showing that all invocations have been successful.

MagicSession

This example demonstrates the usage of "magic sessions" or similarly called implicit correlation¹. ODE supports implicit correlation between two processes or with other services using a session-based protocol. So you don't need to provide any correlation mechanism to establish a stateful interaction. After deployment, start a process with the command:

- Microsoft Windows (XP, Vista, 7)

```
>> ..bin\sendsoap.bat http://localhost:8080/ode/processes/MSMainExecuteService
.MagicSession\testRequest.soap
```

- GNU/Linux (Ubuntu 10.04)

```
>>./bin/sendsoap http://localhost:8080/ode/processes/MSMainExecuteService
./MagicSession/testRequest.soap
```

NOTE: Please make sure that you execute the command from the example directory. The response should be an 'OK' SOAP message, showing that all invocations have been successful.

¹ BPEL process instances are stateful: a client interacting with the BPEL engine must identify the particular instance with which it intends to interact in all of its communications. For this purpose BPEL defines a mechanism called correlation, which allows the process designer to specify which parts of an incoming message should be used to identify the target process instance. To simplify things, ODE provides an alternative correlation mechanism called implicit correlation. This mechanism automatically handles correlation through session identifiers. Further details are provided into ODE [13].

3.1.5 Deployment of ODEDeploymentService

In this section we present the steps required for the deployment of the ODEDeploymentService. This service will run on top of Tomcat where ODE has been deployed and will support the deployment of service chains on the ODE engine. The ODEDeployment service receives from the user (i.e. this could be either process designer or anyone interested in deploying a process) as input a URL request providing a zip file, which contains the appropriate description files. The zip file is created by the user. We describe how the user creates an appropriate zip file in section 5.1. The steps for deploying the service are the following:

Microsoft Windows (XP, Vista, 7) and GNU/Linux (Ubuntu 10.04)

1. Open the ODEDeploymentService.war file and configure the WEB-INF\classes\com\s3lab\ode.properties file, contained into the .war. This file contains key-value entries which the user must set into system dependent values.
 - a. **temporaryDirectory**, is a user specified directory that will be used by the ODEDeploymentService to store the downloaded .zip files and to write the log file. This directory must have write permission.
 - b. **deploymentDirectoryODE**, is the user specified path to the ODE process deployment directory (ode/WEB-INF/processes). This directory is used by ODE to store the deployed processes.
2. Copy the ODEDeploymentService.war file to Apache Tomcat's webapps directory. The service will be automatically deployed on Tomcat.

3.2 Process Optimizer

Current implementations on the Process Optimizer component are focusing on the development of the Input Translator sub-component (see Figure 7). More specifically the features of the Input Translator, which are currently under development, include:

- Appropriate parsers of the provided input (i.e. process and service interface specifications) and internal transformation representations.
- Algorithms for supporting the measurement of relevance between concepts and the discovery of related concepts. These algorithms will be part of the Observation Expander.

With regards to the rest of the Process Optimizer sub-components, i.e. Constraint Expression Provider, Process Specification Provider, Interface To Service Catalogue, Planner, Output Provider, Query Generator and BPEL Formulation Engine, these will be integrated in future releases of the Execution Infrastructure. Since Process Optimizer is currently under development it is not included in version 1.0 of the infrastructure.

The reason we decided to start the implementation of the Process Optimizer with the Input Translator is that this sub-component constitutes the main entrance to the Process Optimizer; in other words, input and stimuli collected from the rest of the infrastructure (via the Deployment Service), enters the Process Optimizer via the Input Translator. More specifically, the Input Translator is responsible for handling the input provided by the Composition Portal (which has been collected via the Deployment Service), and for generating the appropriate internal formats that will be used by the Planner.

The input manipulated by the Input Translator consists of service chain specifications as well as service descriptions of all related services. These are described in terms of:

- WS-BPEL for the description of the service chains,

- WSDL and WSML-Lite descriptions for the description of service interfaces.

All these kinds of input are mandatory for the correct (i.e. both semantically and syntactically valid) generation of the necessary internal representations that will be used by the Planner. The parsing and management of this input is facilitated by appropriate middleware that consists of:

- the open-source WS-BPEL Compiler of the ODE Engine [8],
- the WSL4J open-source parser, which is implemented as part of the SOA4ALL EU project²,
- the WSDL4J open-source parser [9].

Additional details on each of the utilized middleware are presented next.

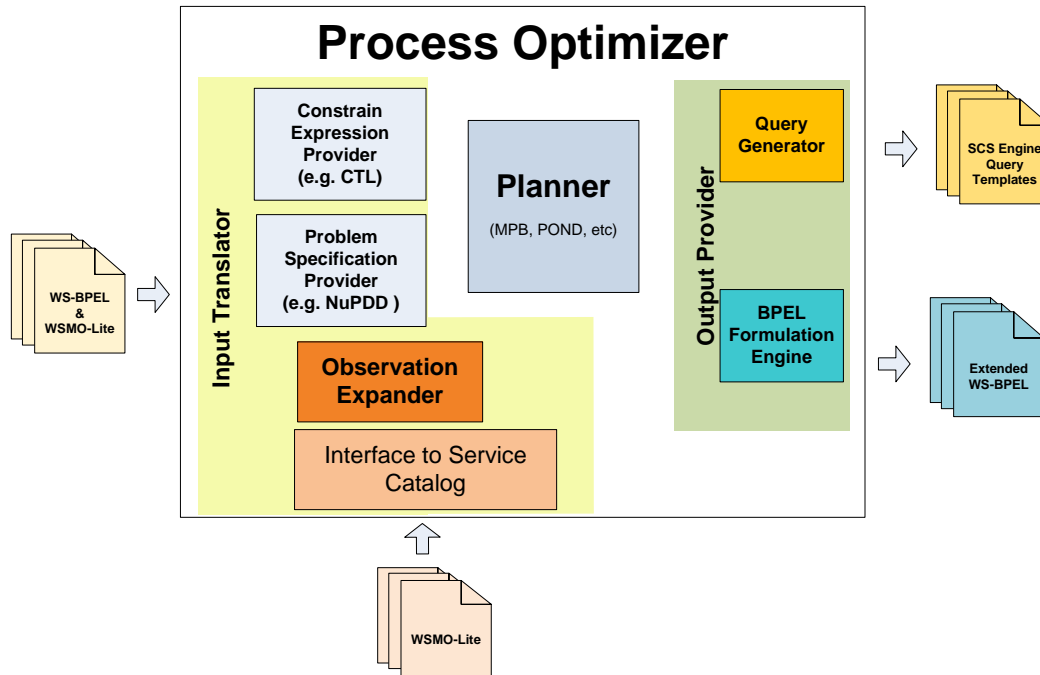


Figure 7 : Process Optimizer Architecture

3.2.1 BPEL Compiler

The BPEL compiler is responsible for parsing the provided BPEL descriptions and transforming them into in-memory Java based representations, which are used by the Input-Translator. In order to ensure the consistency among the Execution Infrastructure components, the BPEL compiler (i.e. ODE version 1.3.4) that is used by Process Optimizer is the one supported by the ODE Engine. The selected compiler supports both BPEL v1.1 and WS-BPEL v2.0 standards for the description of service orchestrations.

The compiler is not offered as a stand-alone library that can be independently downloaded from the ODE Engine's web site. One may acquire it by downloading the source code bundle of the ODE engine and then generate it following the specified process. The source code of the ODE engine is available at the [8].

² SOA4ALL FP7 EU project <http://www.soa4all.eu/>

3.2.2 WSML-Lite Parser

WSL4J is a library implemented as part of the SOA4ALL FP7 project (<http://www.soa4all.eu>) responsible for the parsing and serialization of WSML based service descriptions. This tool is an enhancement of the WSMO4J library incorporating the latest developments on WSML. The Input Translator component of the Process Optimizer uses this tool to facilitate the transformation of provided semantic service descriptions, i.e. defined in WSML-Lite, to the internal in-memory representation.

WSML4J has been provided for use by the Envision Execution Infrastructure via UIBK which is the developer of WSL4J and a collaborator at the SOA4ALL project.

3.2.3 WSDL Parser

WSDL4J (<http://wsdl4j.sourceforge.net/>) is the de-facto parser used for parsing WSDL based specifications in several tools, e.g. ODE Engine. It is implemented as an open-source project that can be easily integrated with other tools and middleware, and extended so as to support the parsing of proprietary WSDL extensions. In Process Optimizer the WSDL4J parser is used along with WSL4J parser to support the transformation of WSDL descriptions to in-memory, Java-based objects. The instantiated in-memory representations are used by the Input Translator for extracting syntactic descriptions/artifacts of the services involved into a service chain.

WSDL4J is an open-source stand-alone tool accessible through [9]. The version used by the Process Optimizer is version 1.6.2.

3.3 Semantic Context Space Engine

The current version of the SCS Engine is not fully implemented and not integrated with the other components, namely Process Optimizer and Service Orchestration Engine. In the following sections we present the frameworks which are used in the current implementation of the SCS Engine and the steps needed to use these frameworks in the implementation.

A critical section in the SCS Engine is the implementation of the Data Manager which in turn consists of the Scope Manager, the Meta-info Based Index Tree, and the Type-based Index Tree. The creation of the Meta-info Based Index Tree requires the knowledge about the semantics of the elements for insertion. These semantics are fed to the SCS Engine through external sources, which in our case are ontologies described in RDFS or WSML. In order to extract information from these ontologies, we need a framework for RDFS files and a framework for WSML files, which will store (if necessary), parse, and query RDFS and respectively WSML data.

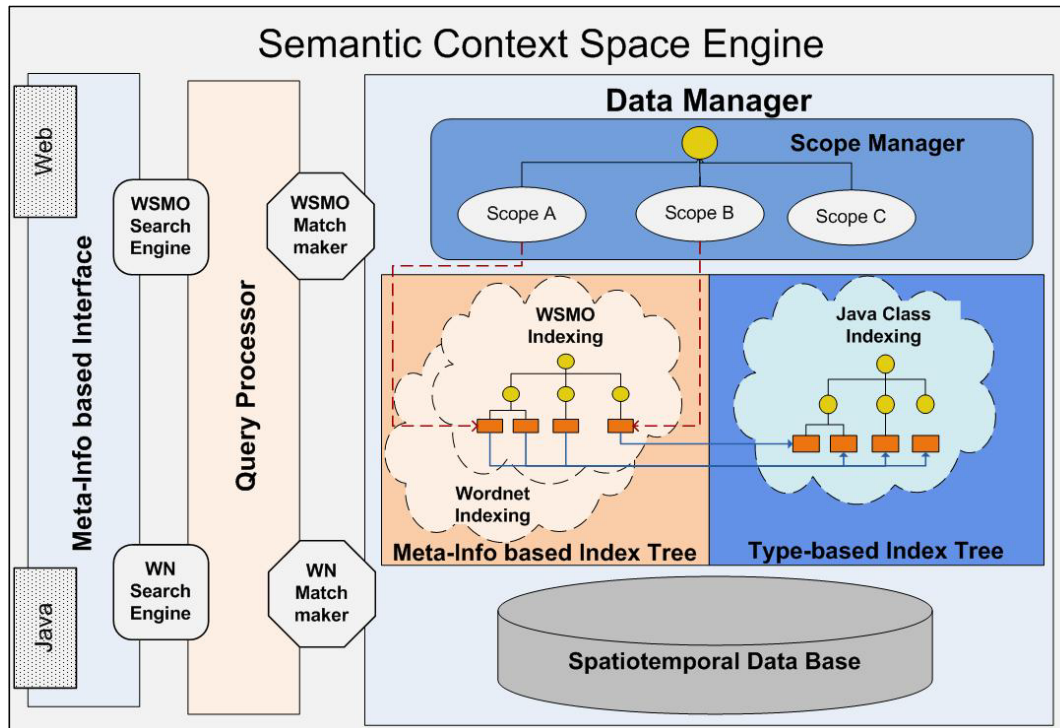


Figure 8 : SCS Engine Architecture

3.3.1 Sesame Repository

For the parsing of RDFS data we use Sesame Repository. Sesame is an open source Java framework for storage and querying of RDF data, consisting of a set of libraries. The framework is fully extensible and configurable with respect to storage mechanisms, inference engines, RDF file formats, query result formats and query languages. Sesame supports SPARQL and SeRQL querying, a memory-based and a disk-based RDF store and RDF Schema inference engines. It also supports most popular RDF file formats and query result formats. Sesame is currently developed as a community project, with Aduna³ as the project leader.

The latest stable version of the Sesame Repository is 2.3.2 and it is available for download through the link [10]. This link provides a number of different files that a user can download, depending on his/hers requirements.

Those not familiar with the Sesame Repository should download the Sesame onejar library under the name "openrdf-sesame-2.3.2-onejar.jar". This file contains all the relevant Sesame libraries. More experienced users, who want more control over what they use, may download the file "openrdf-sesame-(version)-sdk.tar.gz" or "openrdf-sesame-(version)-sdk.zip". In terms of Envision, we decided to use the "openrdf-sesame-2.3.2-onejar.jar" as the user has only to add this file into the classpath.

Before using the Sesame libraries, one important configuration step needs to be taken: the initialization and configuration of a logging framework. Sesame uses the Simple Logging Facade for Java (SLF4J), which is a framework that abstracts the actual logging implementation. SLF4J allows a user of the Sesame framework to plug in his/her own favourite logging implementation at deployment time. SLF4J supports a collection of popular logging implementations such as Java

³ <http://www.aduna-software.com/>

Logging, Apache Commons Logging, and log4j. The SLF4J release packages include bridges to various logging implementations.

In Envision, for the logging implementation we decided to use Apache log4j v1.2 [11]. After downloading and unzipping the file “apache-log4j-1.2.16.tar.gz” or “apache-log4j-1.2.16.zip”, the user should include the appropriate SLF4J-bridge jar-file, namely log4j-1.2.16.jar, in the classpath.

3.3.2 Ontology Representation and Data Integration (ORDI)

The Ontology Representation and Data Integration (ORDI) Framework is an open-source ontology middleware based on Java. Among other things, ORDI provides support for integration of different structured data sources, backward compatibility with the existing RDF specifications and the SPARQL query language, efficient processing and storage of meta-data or context information, and easy management of data from several sources within one and the same repository.

The latest stable version of the ORDI is 0.5 (released on 2009-02-23) [12]. The user should download the file “ordi-0.5-complete.zip”. This zip includes the ORDI implementation files and all the required jar files. As in the case of Sesame Repository, the user only has to add all these jar files into the classpath. After this step, the ORDI framework is ready to use.

4 Supported Users

The list of user roles, as these have been perceived in deliverable D2.1 [3], related to the use of the infrastructure in a direct or indirect manner includes:

- *End-Users*: End-users, i.e. users of the scenario portals, can interact with the infrastructure in two manners: via the indirect invocation of service processes or via the direct subscription of information to the SCS Engine:
 - The first case is the one that will be used by all portal users: whenever someone accesses the scenario portals he/she will -behind the scenes- indirectly create invocation messages, which will be transferred to the infrastructure. The transferred messages will be processed by the corresponding service chain and replies will be generated and returned back to the corresponding invokers.
 - The second type of interaction is a more advanced one which will be available in next releases of the infrastructure. This type of interaction will be made available to the end-users so as to support the adaptation of service chains based on input (i.e. extra information elements) that is directly provided by them. This input, which will have to be semantically annotated, will be collected by the SCS Engine and used for the calculation of appropriate adaptation steps.
- *Designers*: Designers are the kind of users which directly access the execution infrastructure so as to facilitate the deployment and testing of processes.
 - *Process Deployment*: Upon the generation of a process specification in the Composition portal, process designers, will be able to deploy it on the execution infrastructure and specify the required deployment parameters. The infrastructure will return back any emerging error from the deployment process or an acknowledgment of successful deployment.
 - *Process Testing*: Provided that the process has been successfully deployed, a designer will be able to test it using service invocation test tools and/or middleware.

5 Using the Infrastructure

This section describes how the user can facilitate the ODEDeploymentService in order to deploy a new process into the running ODE engine. We provide a deployment example and we show how the user can also send a request to the ODE engine in order to invoke the deployed service.

5.1 Process Deployment

To deploy a process the ODEDeploymentService requires as input a URI to a zip file that has the name of the process. The zip file must be provided by the user, who wants to deploy a new process. The zip file must contain a folder that consists of:

- a BPEL file with the description of the process that will be deployed into the ODE engine. The provision of this file is mandatory.
- a WSDL file describing the public interface of the BPEL process which is offered as a Web Service (WS). The provision of this file is mandatory.
- a number of WSDL files, corresponding to the services used in the process, or a hierarchy of folders containing these files. If the deployed process invokes a WS, then it is obligatory to provide the ODE engine with the WSDL description of the invoked service.

For example if we want to deploy the **testProcess** process we must submit, using **sendsoap**, the following request (contained into **testProcessRequest.soap**):

```
<?xml version="1.0" encoding="utf-8" ?>
<!--
~ This is a deployment request for the testProcess process. This request must be sent
~ to the ODEDeploymentProcess
-->
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:s3l="http://s3lab.com/">
  <!-- test soap message -->
  <soapenv:Header/>
  <soapenv:Body>
    <s3l:deployment>
      <zipFile>file:///C:/examples/testProcess.zip</zipFile>
    </s3l:deployment>
  </soapenv:Body>
</soapenv:Envelope>
```

We invoke the ODEDeploymentService using the usual approach:

- Microsoft Windows (XP, Vista, 7)

```
>>..bin\sendsoap.bat
http://localhost:8080/ODEDeploymentService/EnvDeploymentService
.TestDeploytestProcessRequest.soap
```

- GNU/Linux (Ubuntu 10.04)

```
>> ../bin/sendsoap
http://localhost:8080/ODEDeploymentService/EnvDeploymentService
/TestDeploy/testProcessRequest.soap
```

It should be ensured that the execution of the above command is performed from the example directory. The response will be a SOAP message containing the 'Success' string if the process is deployed successfully or a message containing "Failure" with the appropriate error description.

5.2 Process Invocation

The testProcess that we previously deployed returns the sum of two integers. So if we invoke the testProcess using a request that provides two numbers, the process will return their sum.

For example we will send the request described into *testRequest.soap*

```
<?xml version="1.0" encoding="utf-8" ?>
<!--
    ~ This is a request to the testProcess.
    ~ The testProcess adds the two input numbers.
-->
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:web="http://webservice.simple/">
  <soapenv:Header/>
  <soapenv:Body>
    <web:Add2Integers>
      <!-- parameter 1 -->
      <Number1>7</Number1>
      <!-- parameter 2 -->
      <Number2>13</Number2>
    </web:Add2Integers>
  </soapenv:Body>
</soapenv:Envelope>
```

We invoke the deployed testProcess using the usual approach:

- Microsoft Windows (XP, Vista, 7)

```
>> ../bin/sendsoap.bat http://localhost:8080/ode/processes/testProcess
.\TestProcess\testRequest.soap
```

- GNU/Linux (Ubuntu 10.04)

```
>> ../bin/sendsoap http://localhost:8080/ode/processes/testProcess
/TestProcess/testRequest.soap
```

The correct returned value will be the sum of the two input numbers i.e. 20.

6 Summary

The Execution Infrastructure constitutes one of the main components of the Envision platform providing the required support for the execution, data-driven adaptation and mediation of service

envision

chains. This component can be thought of as the ‘heart’ of the platform bringing life to the service chains designed by the Composition portal and used by the scenario portals.

It was decided that the infrastructure will be designed and implemented in an incremental manner. Specifically, it has been agreed to support the deployment and execution of service processes right from ver. 1.0 of the Execution Infrastructure. Additional features on data-driven adaptation and semantic based service mediation will be offered in following versions.

The main goal of this document was to provide an overview of the features that are included in the version 1.0 of the infrastructure and to provide the necessary guidelines on how they can be installed and used. In this context, we have provided details on the deployment process of the ODE engine on a single server node as well as on the manners one may use so to invoke running processes.

The current status of the components comprising the infrastructure is summarized into the following table (see Table 1).

Table 1: Execution Infrastructure Implementation Status

Component	Currently Available	Under Development	Future Work
Service Orchestration Engine	<ul style="list-style-type: none"> • Single instance of Orchestration Engine • Process Deployment Service 	<ul style="list-style-type: none"> • Peer-to-Peer infrastructure of the Orchestration Engine 	<ul style="list-style-type: none"> • Peer-to-Peer deployment of the Engine • Integration with the rest of the Execution Infrastructure components • OGC service interface front-end
Semantic Context Space Engine	<ul style="list-style-type: none"> • Preliminary version supporting WordNet based meta-data • Java-based interface for the interaction with other components and external data sources 	<ul style="list-style-type: none"> • Support for the annotation of collected information using RDFS and WSML based ontologies 	<ul style="list-style-type: none"> • Web-based interface for the integration with the rest of the Execution Infrastructure components • Support for the use of spatiotemporal meta-data
Process Optimizer		<ul style="list-style-type: none"> • Input Translator supporting the transformation of provided service & process descriptions into internal representations • Implementation of Observation Expander based on semantic based relevance measurements 	<ul style="list-style-type: none"> • Integration of Input Translator with Semantic Service Catalogue • Integration with Process Planner • Implementation of Output Provider
Data Mediation Engine	<ul style="list-style-type: none"> • XSLT based data-mediations 		<ul style="list-style-type: none"> • Semantic based data mediation

Our preliminary investigation of the provided outcomes unveiled that the complexity of the pilot application scenarios in terms of required control flow and data flow structures is from medium to low, e.g. comprising sequence or flow activities without control structures such as if, switch, etc. At the same time their needs in computation resources and data volume exchanges are quite

high. Therefore, the use of centralized (single server node) architectural style for the deployment of the Orchestration Engine is rather limiting. In addition, the selection of the most appropriate architectural style for the distribution of Orchestration Engine is a crucial step that should be thoroughly investigated and associated to the needs/properties of the pilot cases.

Another important aspect that has a clear effect on the performance characteristics of the Orchestration Engine is related to the complexity of the data transformations required by each service process. Although, BPEL inherently supports the execution of simple copy assignments/mappings the use of data types and more complex mappings require the use of XSL transformations that have an effect on the process execution time. Considering also the volume of the exchanged data this time may vary significantly.

To conclude, we should note that the implementation and integration of the additional features missing from this first release will clearly improve the performance of the provided infrastructure. It is expected that both the distribution of the Orchestration Engine and the data-driven adaptation of service processes will reduce the required execution time.

7 References

- [1] Envision Consortium, "Annex I- Description of Work," ENVISION project FP7 - Contract no 249120, DoW 2009.
- [2] G. Athanasopoulos, Y. Pogas, P. Kouki, M. Pantazoglou, Ioan Tomas A. Tsalgatidou, "D6.1 - ENVISION Adaptive Execution Infrastructure – Architecture Specification, ver.1.1," Envision project, Deliverable 2010.
- [3] J. Langlois, F. Tertre, F. Husson, and P. Maue, "D2.1: Environmental semantic Web Portal Architecture Specification," Envision project no 249120, Deliverable 2010.
- [4] Oracle. Java SE at a Glance. [Online]. <http://java.sun.com/j2se>
- [5] Apache org. Apache Tomcat 6.0: Tomcat setup. [Online]. <http://tomcat.apache.org/tomcat-6.0-doc/setup.html>
- [6] Apache org. Apache Tomcat: Download. [Online]. <http://tomcat.apache.org/download-60.cgi>
- [7] Ubuntu Documentation Team. Ubuntu Documentation. [Online]. <https://help.ubuntu.com/10.04/serverguide/C/tomcat.html>
- [8] Apache org. Apache ODE: Getting Ode. [Online]. <http://ode.apache.org/getting-ode.html>
- [9] Apache org. Apache Ode: Project Site. [Online]. <http://ode.apache.org>
- [10] WSDL4J Community. WSDL4J SourceForge Project. [Online]. <http://wsdl4j.sourceforge.net/>
- [11] Sesame Community. Sesame SourceForge Project. [Online]. <http://sourceforge.net/projects/sesame/>
- [12] Apache org. Apache Log4j Project. [Online]. <http://logging.apache.org/log4j/>
- [13] Ordi Community. Ordi SourceForge Project. [Online]. <http://sourceforge.net/projects/ordi>
- [14] Apache org. Apache ODE: Implicit Correlations. [Online]. <http://ode.apache.org/implicit-correlations.html>