



envision
environmental services infrastructure with ontologies

Deliverable D6.3:

**ENVISION Adaptive Execution Infrastructure Version 2.0 -
Improved Development & User Guide**

Date:	01 February 2012
Author(s):	A. Tsalgatidou, G. Athanasopoulos, P. Kouki, M. Pantazoglou, Y. Pogkas (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), Henry Michels (UOM)
Contributor(s):	Ioan Toma (UIBK)

Version history

Version	Date	Description
0.1	24/10/2011	Initial table of contents
0.2	25/10/2011	Updated table of contents
0.3	16/11/2011	Initial input
0.4	01/12/2011	Additional input to various sections
0.5	05/12/2011	Changes to interface sections
0.6	20/12/2011	Added UML diagrams and SOAP message examples
0.7	23/12/2011	Release for internal review
0.8	16/01/2012	Incorporated comments from internal reviewers
0.9	31/01/2012	Version approved by the technical coordinator
1.0	01/02/2012	Final version approved by the project coordinator

Executive Summary

The ENVISION Adaptive Execution Infrastructure constitutes a fundamental runtime component of the overall ENVISION platform, and is responsible for the execution of environmental models that are implemented as *WS-BPEL (Web Services Business Process Execution Language)* [1] processes, supporting their data-driven adaptation as well as the semantic-based mediation among the constituent services. All those features are composed to provide the operational core of the ENVISION runtime environment and support the functionality of the ENVISION Portal.

The goal of this deliverable is to report on the developments accomplished during the second year of the ENVISION project, giving an account of the features implemented in the second version of the ENVISION Adaptive Execution Infrastructure, and providing a guide with details on the installation and use of its main components. The deliverable intentionally does not incrementally build upon its predecessor, ENVISION deliverable D6.2 [D6.2]; rather, it has been written from scratch to better present the many significant developments that have been encompassed in the new version of the accompanying software prototype.

Table of Contents

1	Introduction	6
2	Architecture Overview	7
2.1	Components and Functionality	7
2.1.1	Deployment Service	7
2.1.2	Semantic Context Space Engine	8
2.1.3	Data Mediation Engine	9
2.1.4	Service Orchestration Engine	9
2.2	Interfaces	11
2.2.1	Deployment Interface	11
2.2.2	Data Acquisition Interface	12
2.2.3	Execution & Monitoring Interface	14
3	Installation Guide	17
3.1	Service Orchestration Engine	17
3.1.1	Prerequisites	18
3.1.2	Installation	18
3.1.3	Configuration	19
3.2	Deployment Service	21
3.2.1	Prerequisites	21
3.2.2	Installation	21
3.2.3	Configuration	22
3.3	Semantic Context Space Engine	23
3.3.1	Prerequisites	23
3.3.2	Installation	23
3.3.3	Configuration	23
3.4	Data Mediation Engine	25
3.4.1	Prerequisites	25
3.4.2	Installation	25
3.4.3	Configuration	25
4	User Guide	25
4.1	Target Users	26
4.2	Deployment of the Drought Risk Assessment Model	26
4.3	Execution of the Drought Risk Assessment Model	26
4.4	Monitoring of the Drought Risk Assessment Model	27
4.5	Undeploying the Drought Risk Assessment Model	29
5	Conclusions	32
	References to ENVISION Deliverables	33
	References	34
Appendix A	Data-driven Adaptation of Environmental Models	36
A.1	Introduction	36
A.2	Process Adaptation	36
A.3	Observation Variable Identification	37
A.4	Observation and Action Expansion	38
A.4.1	Observation Concept Expansion	38
A.4.2	Action Expansion	39
A.4.3	Service Query Formulation	41
A.5	Observation and Action Set Definition	42

List of Figures

Figure 1	The ENVISION Adaptive Execution Infrastructure	7
Figure 2	Core Elements of the Information Model of the SCS Engine	8
Figure 3	The P2P Engine	10
Figure 4	The Deployment Interface	11
Figure 5	The Data Acquisition Interface	12
Figure 6	The Monitoring Interface	15
Figure 7	Deployment of the ENVISION Adaptive Execution Infrastructure	18
Figure 8	Web page with information produced by the P2P Engine Servlet	20
Figure 9	The drought risk assessment model	26
Figure 10	Execution of the drought risk assessment model	28
Figure 11	Retrieving the list of process instances through the Monitoring Interface	30
Figure 12	Monitoring intermediate results	31
Figure 13	Communication channel between a process and external service providers	37
Figure 14	Observation and Action Expansion Process	38

1 Introduction

This deliverable presents the features that have been implemented in version 2.0 of the ENVISION Adaptive Execution Infrastructure. Furthermore, it provides a user guide to assist designers and end-users in deploying, executing, and monitoring environmental models that are implemented as WS-BPEL processes according to the ENVISION approach. The previous version of the infrastructure, documented in ENVISION deliverable D6.2, mainly featured preliminary implementations of the various components and interfaces. Building on those initial prototypes, this version incorporates significant contributions. Emphasis was particularly given on the components that support the deployment, distributed execution, and monitoring of environmental models based on WS-BPEL.

More specifically, compared to version 1.0, this version of the ENVISION Adaptive Execution Infrastructure contributes updates to the following components:

- **Process Optimizer:** This component has been further developed in order to implement the transformation of WSML (Web Service Modeling Language) [5], WSDL (Web Service Description Language) [4] and WS-BPEL descriptions to the internal state transition model representations. Algorithms for the calculation of semantically related observations and the discovery of additional external services have been developed. Further, the component has been integrated with the ENVISION Semantic Catalogue [D5.4] for the discovery of additional services. It should be noted that, in the updated architecture of the ENVISION Adaptive Execution Infrastructure, the Process Optimizer has become an internal component of the Deployment Service.
- **Deployment Service:** This component has undergone changes in order to improve its stability. From an external point of view, it has become operational and fully integrated with the ENVISION portal's Resource Portlet, allowing the deployment of environmental model bundles to the Service Orchestration Engine. In its final version, it will be also integrated with the Process Optimizer.
- **Service Orchestration Engine:** The centralized WS-BPEL engine that is based on Apache ODE has been updated from version 1.3.4 to version 1.3.5, in order to benefit from the many performance improvements and bug fixes. Furthermore, the P2P infrastructure for the distributed execution of WS-BPEL processes has been implemented and integrated with the centralized WS-BPEL engine. In addition, a new distributed monitoring mechanism has been developed to address the ENVISION user requirements.
- **Semantic Context Space (SCS) Engine:** Extensions catering for the annotation of collected information using RDFS and WSML-based ontologies have been completed. Also, the additional extensions catering for the spatiotemporal annotation of the collected information have been developed. Finally, aside from the existing Java RMI-based interface, a SOAP-based interface for data acquisition has been released. It should be noted though that the SCS Engine will be integrated with the rest of the components of the execution infrastructure in the final version, 3.0.
- **Data Mediation Engine:** The development of this component has been completed as well as its delivery as a SOAP-based Web service. Like the SCS Engine though, its full integration with the rest of the ENVISION Adaptive Execution Infrastructure will be accomplished during the last year of the project.

The remainder of the deliverable is structured as follows: In Section 2, we provide a brief description of the updated architecture and the comprising components of the ENVISION Adaptive Execution Infrastructure. Moving on to Section 3, a set of detailed guidelines are given to facilitate the installation of the provided components. A number of example scenarios are given in Section 4 to demonstrate and explain the usage of the ENVISION Adaptive Execution Infrastructure by environmental model designers and end-users. Finally, the deliverable is concluded in Section 5 with an outlook on the work that remains to be carried out in Work Package 6 until the end of the project.

2 Architecture Overview

This section provides an updated architectural overview of the ENVISION Adaptive Execution Infrastructure in terms of its main components, functionality, and interfaces. The goal is to provide the administrator and user of the execution infrastructure with the necessary knowledge in order to better understand the purposes this infrastructure serves, and to be able to easily follow the installation and usage instructions that are given in the succeeding sections. Descriptions have been purposely kept at a high level; hence, for more technical details, the reader may refer to ENVISION deliverable D6.1 [D6.1].

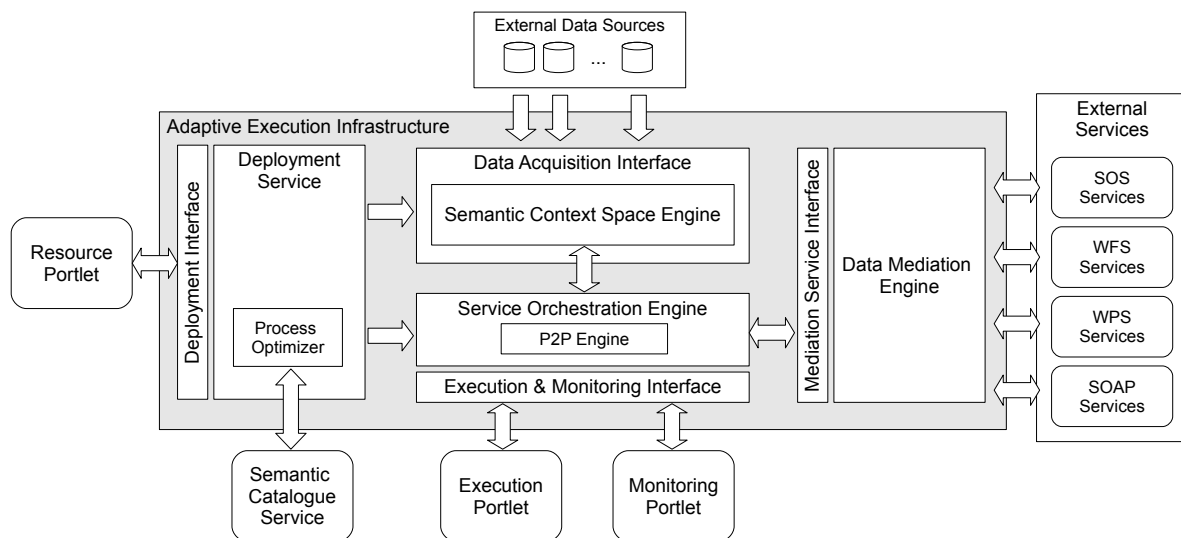


Figure 1: Architecture of the ENVISION Adaptive Execution Infrastructure

2.1 Components and Functionality

Figure 1 illustrates a high-level view of the architecture of the ENVISION Adaptive Execution Infrastructure, which comprises a set of high-level components, namely the **Deployment Service**, the **Semantic Context Space (SCS) Engine**, the **Data Mediation Engine**, and the **Service Orchestration Engine**. These components enable the deployment and adaptive execution of environmental models as WS-BPEL processes, while also catering for their monitoring in a distributed and scalable manner. The following paragraphs provide more details on the purposes and functionality of each one of these main components.

2.1.1 Deployment Service

The Deployment Service is the entry point for the deployment of environmental models as WS-BPEL processes to the ENVISION Adaptive Execution Infrastructure. In a typical usage scenario, this component accepts a bundle from the ENVISION Resource Portlet (for more details on this Portlet, please refer to ENVISION deliverable D2.3 [D2.3]), which contains the WS-BPEL process file and all its accompanying artefacts (WSDL [4] documents, XSD [17] and XSLT [18] files, etc.). The contents of the submitted bundle are processed by an internal component of the Deployment Service called **Process Optimizer**, which performs all necessary work to expand and render the originally submitted WS-BPEL process adaptable. The Process Optimizer is therefore critical for the realisation of the data-driven adaptation approach that ENVISION follows in the execution of environmental models. Briefly¹, its main objectives are to

¹The approach taken by the Process Optimizer towards data-driven adaptation of BPEL processes is elaborated in Appendix A

- annotate the provided WS-BPEL processes with extension points that are evaluated at runtime, and
- accommodate process adaptation based on the exploitation of available information elements, which are external to the process execution context.

These objectives are met by the Process Optimizer by specifying

- the set of information elements, which are relevant to a given environmental model, and should be pushed by the SCS Engine to its instances upon execution, and
- the adaptation steps (equivalently referred to as *plans*) that should be performed upon the discovery of such information at runtime.

As it was previously mentioned, the outcome of this processing is an expanded WS-BPEL process definition which is dispatched to the Service Orchestration Engine. In turn, the latter binds the deployed WS-BPEL process to a unique Web service endpoint address. Hence, in compliance to standards and common practices, all environmental models that are deployed to the ENVISION Adaptive Execution Infrastructure can be conveniently invoked as standard SOAP [8] Web services.

The Deployment Service also supports the un-deployment of environmental models that were previously deployed but are no longer used, or they need to be substituted. Such functionality is important for reasons of maintenance, versioning, and helps keeping the ENVISION Adaptive Execution Infrastructure clean of unwanted/unnecessary models.

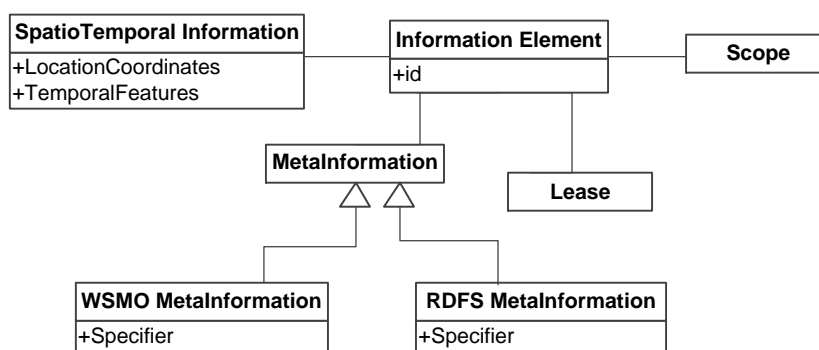


Figure 2: Core Elements of the Information Model of the SCS Engine

2.1.2 Semantic Context Space Engine

The main goal of the Semantic Context Space (SCS) Engine is to provide an open mechanism for data acquisition. Specifically, this mechanism supports the collection and sharing of *external*² information elements, which are basically structured data accompanied by appropriate semantic annotations. Each information entity stored in the SCS Engine should be in a specific form, as illustrated in Figure 2. In particular, the main attributes of an information entity are: (i) the unique identifier of each information element; (ii) a class named *Lease* that adds temporal properties, as it represents a fixed period of time in which the information element is considered to be valid; (iii) a class called *SpatioTemporalInformation*, which keeps the spatiotemporal characteristics of the information element; (iv) a *Scope*, which indicates the category in which the information element belongs to; and (v) a class called *MetalInformation*, which is responsible for keeping all the semantic information related to the information element. The *MetalInformation* is annotated with semantic attributes which can be given in RDFS or WSMML.

²We use this term to distinguish between data that are produced/consumed within the context of a given process instance, and data that, although being produced by non-anticipated, external entities, are related to that process instance.

The acquisition mechanism is independent of the metadata primitives used for the annotation of information elements, and supports their logical organisation into groups, also referred to as *scopes*. The SCS Engine provides its connected clients, i.e. external data sources, with a basic set of operations, which support the efficient writing, grouping, and retrieving of information elements. The latter can be enhanced with the addition of semantic, spatial, and temporal metadata annotations.

A more detailed description of the purposes and intended functionality of the SCS Engine is given in ENVISION deliverable D6.1 [D6.1]. Concluding, we would like to note that, although the SCS Engine is fully implemented in the second version of the ENVISION Adaptive Execution Infrastructure, it has not yet been integrated with the Process Optimizer and the Service Orchestration Engine.

2.1.3 Data Mediation Engine

The Data Mediation Engine is the sub-component of the ENVISION Adaptive Execution Infrastructure that is responsible for resolving data heterogeneity issues that occur in a chain of services invocation. More precisely, the Data Mediation Engine, takes as input:

- a mapping m between two ontologies O_1 and O_2 that are used to annotate output of service S_1 , respectively input of S_2 , where there is composition linking the output of S_1 to the input of S_2 ,
- a set of instances according to O_1 that are produced (output) by S_1 ,

and delivers as output:

- a set of instances according to O_2 that will be consumed (input) by S_2 .

The O_1 instances are mapped to O_2 instances according to the mapping m that is created at design time. The Data Mediation Engine component is deployable as a Web service and can be invoked by providing the input mentioned before. Specifically, the exposed WSDL interface comprises one operation

The Data Mediation Engine functionality is exposed as a WSDL service³. This service comprises one operation called **mapOntology**, which takes as input parameter a **mapOntologyRequest** while it returns a **mapOntologyResponse**. A **mapOntologyRequest** includes:

- a **sourceOntology** containing the source ontology plus a set of instances that need to be transformed by the Data Mediation Engine;
- a **targetOntology** containing the target ontology;
- a **mappingDocument** containing the mappings between the source and target ontology;

In turn, a **mapOntologyResponse** includes:

- a string representation of the transformed instances according to the target ontology.

It should be noted that, although the functionality is fully implemented, still the Data Mediation Engine has not yet been integrated with the other components of the ENVISION Adaptive Execution Infrastructure. This will be done during the third year of the project.

2.1.4 Service Orchestration Engine

The Service Orchestration Engine is the central component of the ENVISION Adaptive Execution Infrastructure, and is responsible for the distributed execution and monitoring of the deployed environmental models. In doing so, the Service Orchestration Engine extends a centralised WS-BPEL engine by deploying some of its internal components on a scalable peer-to-peer infrastructure, which is dubbed **P2P Engine** and was implemented with the use of the JXTA [7] technology.

³The WSDL description of the Data Mediation Engine can be checked online at: <http://sesa.sti2.at:8080/MappingWebProject/services/Mapper?wsdl>

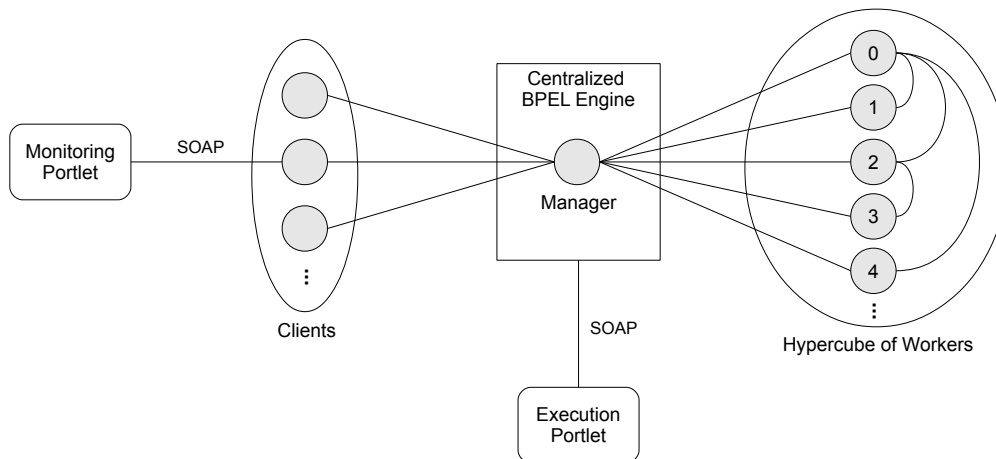


Figure 3: Architecture of the P2P Engine and interactions with ENVISION portlets

Figure 3 illustrates the overall architecture of the P2P Engine, which consists of three different types of nodes serving complementary purposes:

- The P2P Engine **Manager** is embedded within the centralised WS-BPEL engine, and is responsible for (i) dispatching executable parts of a WS-BPEL process, such as assign or invoke activities, for distributed execution by Worker nodes, and (ii) sending meta-information about WS-BPEL process instances and their variables to registered Client nodes.
- The P2P Engine **Workers** are responsible for the distributed execution of specific tasks in a WS-BPEL process, such as the assign and invoke activities, which have been identified as the source of performance bottlenecks in centralised WS-BPEL engines. Further, they also take over the handling and storage of data variables, thereby alleviating the WS-BPEL engine from the need to store a potentially huge volume of data that are produced and/or consumed upon execution of multiple environmental model instances. The P2P Engine Worker nodes are organised in a Hypercube [15] peer-to-peer topology, which ensures efficient broadcasting and failure recovery. Furthermore, to optimise the process execution performance and scalability, the Manager always assigns variables and/or activities to the *Least Recently Used (LRU)* Workers, thereby ensuring a balanced distribution of workload within the P2P Engine.
- The P2P Engine **Clients** are responsible for monitoring the execution of the deployed WS-BPEL processes by the Service Orchestration Engine. The Monitoring Portlet makes use of the Web Service interface exposed by the Client node, in order to send monitor requests about specific process instances and/or variables. After its deployment and initialisation, the Client node registers to the P2P Engine Manager, enabling the latter to forward to the former meta-information about new process instances and variable assignments. This information is then used by the Client to make direct calls to specific P2P Engine Workers that are responsible for particular activities or variables.

Upon execution of a WS-BPEL process, the Service Orchestration Engine is able to retrieve external data that are relevant to that process, and thus may trigger its adaptation. Such data is retrieved from the SCS Engine by means of the subscribe and notify methods of the Data Acquisition Interface (see Figure 1). Also, if a WS-BPEL process requires data mediation upon execution, the Data Mediation Engine is accessed through the Mediation Service Interface, in order to carry out the necessary data transformations⁴.

⁴These capabilities will become available to the Service Orchestration Engine in the final version of the ENVISION Adaptive Execution Infrastructure, where all internal components will be well-integrated.

2.2 Interfaces

The ENVISION Adaptive Execution Infrastructure defines a set of interfaces, which are intended to facilitate (i) the integration with the remainder components of the ENVISION platform, and (ii) the use of the infrastructure by the ENVISION pilot applications. In the following paragraphs, we go through each one of these interfaces discussing their purpose, functionality, expected inputs, and produced outputs.

2.2.1 Deployment Interface

The **Deployment Interface** provides to external entities/actors such as the ENVISION Resource Portlet remote access to the Deployment Service, in order to deploy/un-deploy environmental models to/from the ENVISION Adaptive Execution Infrastructure. These functionalities are exposed as a SOAP-based Web service comprising two synchronous (i.e. blocking) operations.

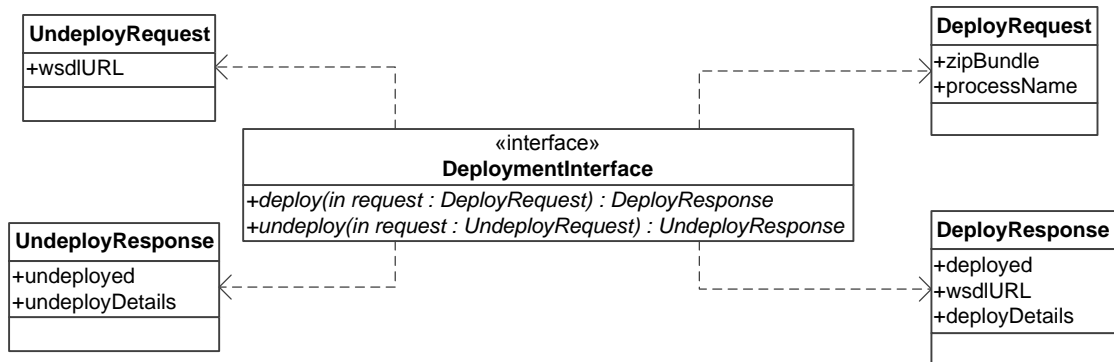


Figure 4: The Deployment Interface

Figure 4 illustrates the Deployment Interface with the use of a UML class diagram, while the provided operations are described in Table 2.1.

Table 2.1: Operations provided by the Deployment Interface

Operation	Input Parameters	Output Parameters
<p>deploy. Used for the deployment of environmental models implemented as WS-BPEL processes to the ENVISION Adaptive Execution Infrastructure.</p>	<p>zipBundle (<i>required</i>). A byte array containing the documents related to the environmental model, i.e. all necessary WS-BPEL, WSDL, XSD, and XSLT files.</p> <p>processName (<i>required</i>). The name of the process which must be equal to the <code>name</code> attribute of the root <code>process</code> element within the WS-BPEL file.</p>	<p>deployed. Indicates whether the deployment was successful or not.</p> <p>wsdlURL. The URL of the WSDL file that can be used for the execution of the deployed WS-BPEL process.</p> <p>deployDetails. Textual description providing additional details about the deployment.</p>
<p>undeploy. Allows the undeployment of environmental models that have been deployed to the ENVISION Adaptive Execution Infrastructure.</p>	<p>wsdlURL (<i>required</i>). The URL of the WSDL file that corresponds to the deployed WS-BPEL process and uniquely identifies the process within the execution infrastructure.</p>	<p>undeployed. Indicates whether the undeployment was successful or not.</p> <p>undeployDetails. Textual description providing additional details about the undeployment.</p>

2.2.2 Data Acquisition Interface

The **Data Acquisition Interface** provides a set of operations, which allow users to read, write, and take information from/to the Semantic Context Space Engine. In addition to the actual information, the provided operations take into account its semantic, spatial, and temporal annotations. The Data Acquisition Interface is implemented both as an RMI-based⁵ and SOAP-based service. Furthermore, a persistent query-like mechanism is provided to support the asynchronous interaction between the SCS Engine and its clients. According to this mechanism, a client submits her query, and the SCS Engine immediately returns a unique identifier. Afterwards, the client can use that identifier to retrieve data that match the originally submitted query.

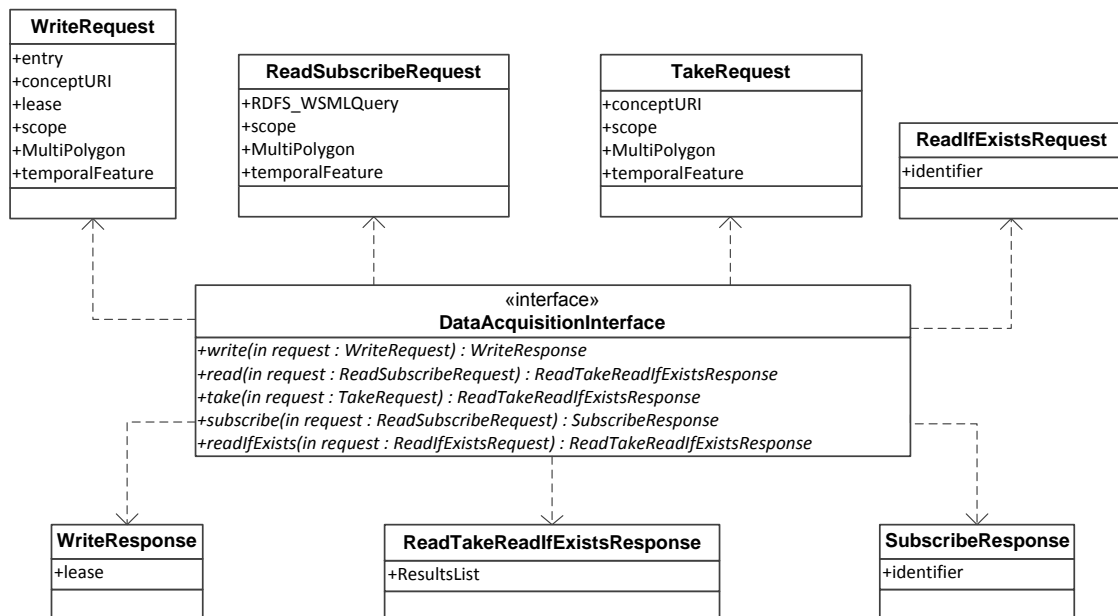


Figure 5: The Data Acquisition Interface

Figure 5 illustrates the Data Acquisition Interface with the use of a UML class diagram, while the provided operations are described in Table 2.2.

⁵RMI, <http://www.oracle.com/technetwork/java/javase/tech/index-jsp-136424.html>

Table 2.2: Operations provided by the Data Acquisition Interface

Operation	Input Parameters	Output Parameters
<p>write. Writes a semantically annotated entry in the SCS Engine. The SCS Engine can also take into account the spatiotemporal characteristics that accompany the given entry, as well as the possible scope, which this entry belongs to.</p>	<p>entry (<i>required</i>). The actual information to be inserted in the SCS Engine.</p> <p>conceptURI (<i>required</i>). The URI pointing to an ontology concept.</p> <p>lease (<i>required</i>). The requested lifetime of the inserted information in the Semantic Context Space Engine, measured in milliseconds.</p> <p>scope (<i>optional</i>). The belonging scope of the inserted information entry.</p> <p>MultiPolygon⁶ (<i>optional</i>). The exact coordinates of the location where the inserted information belongs to.</p> <p>temporalFeature (<i>optional</i>). The temporal features of the inserted information comprising three elements: (i) an indicator of the moment when the information element begins to be valid; (ii) an indicator of the moment when the information element stops to be valid; and (iii) the corresponding time zone.</p>	<p>lease. The lifetime of the inserted entry in the SCS Engine, measured in milliseconds.</p>
<p>read. Retrieves from the SCS Engine information elements which match the specified semantic criteria. It can also consider spatiotemporal criteria and criteria related to the belonging scope of the requested information.[6pt]</p>	<p>RDFS_WSMLQuery (<i>required</i>). Indicates a query with the user-specified search criteria, which consists of: (i) the URI of the ontology concept that semantically describes the desired information; and (ii) the minimum semantic degree of match threshold.</p> <p>scope (<i>optional</i>). The scope within the SCS Engine, in which the read operation will be performed.</p> <p>MultiPolygon (<i>optional</i>). The exact coordinates of a specific location, within which relative entries should be looked up for.</p> <p>temporalFeature (<i>optional</i>). The temporal features of the requested information (as described in (<i>write</i>) operation).</p>	<p>ResultsList. A list of all the matching results. Each entry in the results list has the following four attributes: (i) the information stored in the SCS Engine; (ii) its semantic degree of match with respect to the query; (iii) the distance in terms of location between the given spatial criteria and the returned result; and (iv) the overlapping period between the given temporal criteria and the returned result. The first two attributes are never set to null, in contrast to the latter two, the values of which depend on whether spatial and/or temporal criteria were specified in the submitted request, or not.</p>
<p>take. Similar to the read operation, with the difference that the retrieved information elements are at the same time deleted from the SCS Engine.</p>	<p>conceptURI (<i>required</i>). An ontology concept URI.</p> <p>scope (<i>optional</i>). The scope in which the take operation will be performed.</p> <p>MultiPolygon (<i>optional</i>). The same as in the (<i>read</i>) operation.</p> <p>temporalFeature (<i>optional</i>). The same as in the <i>read</i> operation.</p>	<p>ResultsList. A list of all the matching results, as described in <i>read</i> operation.</p>

⁶For more information, see <http://postgis.refractorions.net/documentation/javadoc/org/postgis/MultiPolygon.html>

<p>subscribe. Similar to the read operation, with the difference that the submitted query is not executed; rather, it is stored in the SCS Engine.</p>	<p>RDFS_WSMLQuery (<i>required</i>). Described in the <i>read</i> operation.</p> <p>scope (<i>optional</i>). Described in the <i>read</i> operation.</p> <p>MultiPolygon (<i>optional</i>). Described in the <i>read</i> operation.</p> <p>temporalFeature (<i>optional</i>). Described in the <i>read</i> operation.</p>	<p>identifier. Indicates a unique identifier which corresponds to the submitted criteria and can be used by the user later on.</p>
<p>readIfExists. Complements the subscribe operation by allowing the user to request for the execution of a previously stored query.</p>	<p>identifier (<i>required</i>). The passed identifier.</p>	<p>ResultsList. Described in the <i>read</i> operation.</p>

2.2.3 Execution & Monitoring Interface

The ENVISION Adaptive Execution Infrastructure relies on the centralised WS-BPEL engine (as implemented by the Apache ODE project⁷) for the execution of WS-BPEL processes. Thus, the **Execution Interface** is defined by and consists of all the WSDL interfaces of the WS-BPEL processes that have been deployed to the Service Orchestration Engine. In general, each deployed WS-BPEL process offers at least one Web service operation, which is exposed to the interested clients via a WSDL interface. This way, any deployed environmental model represented as WS-BPEL process can be invoked as a Web service, with the use of any SOAP-based, Web service client.

On the other hand, a new monitoring service and its respective interface were implemented to address the specific needs of the ENVISION project. We decided not to be based on the management API provided by Apache ODE 1.3.5 for the following reasons:

1. The Apache ODE 1.3.5 Management API is rather complicated compared to the simple monitoring requirements of the ENVISION users.
2. There was an explicit requirement for support of process instance monitoring at the user level, i.e. allowing a user to monitor only her own process instances. Such feature is not provided by the existing API.
3. The implementation of the Apache ODE 1.3.5 Management API is centralised, and as such it would likely introduce performance issues when dealing with large numbers of requests, large variable values, etc.
4. We would be forced to consume a lot more resources to customise the existing implementation and adapt it to the distributed architecture of the Service Orchestration Engine.

Specifically, the monitoring activities have been implemented by leveraging the services offered by the P2P Engine Client, and are conveniently exposed as a SOAP-based Web service. Thus, the **Monitoring Interface** provides users with a number of synchronous (i.e. blocking) operations, allowing them to keep track of their process instances, as well as to retrieve intermediate results for long-running processes.

A UML representation of the Monitoring Interface and its operations with their inputs and outputs is displayed in Figure 6, while Table 2.3 provides more detailed explanations.

⁷<http://ode.apache.org>

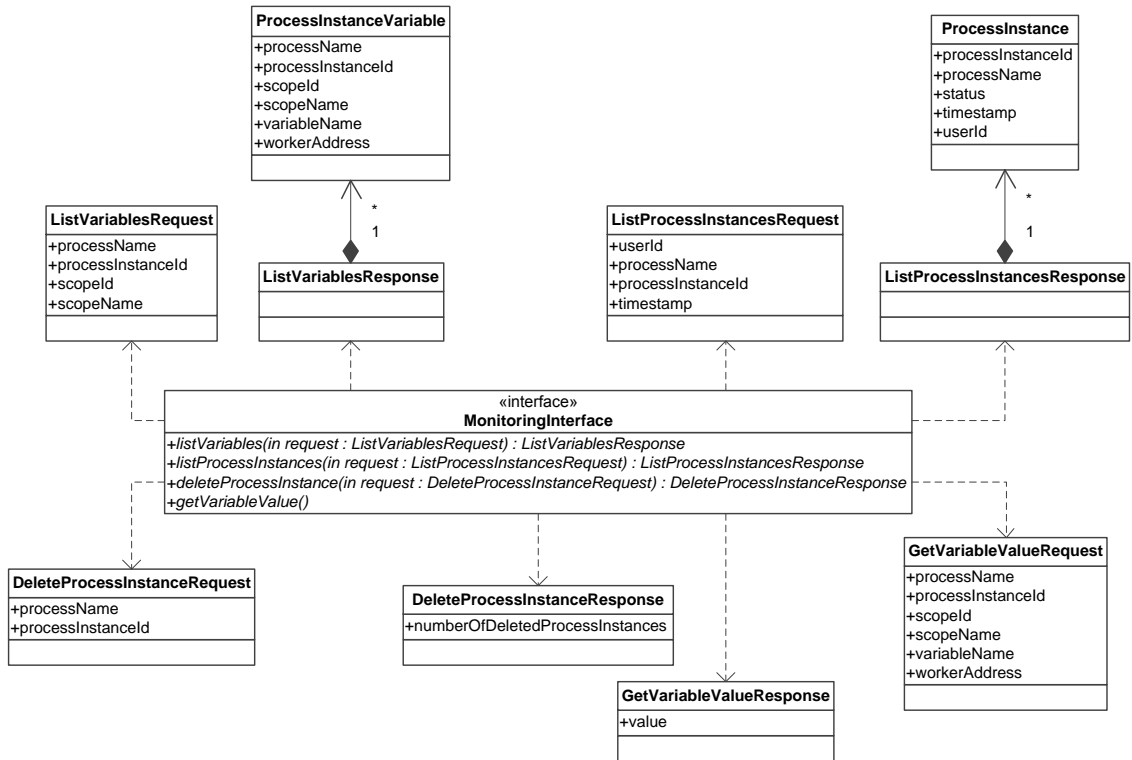


Figure 6: The Monitoring Interface

Table 2.3: Operations provided by the Monitoring Interface

Operation	Input Parameters	Output Parameters
<p>listProcessInstances. Allows users to retrieve information about the process instances.</p>	<p>userId (<i>required</i>). The user identifier, which is used to filter process instances that belong to a specific user. It can also contain wildcards like * (<i>for multiple character matching</i>) and ? (<i>for single character matching</i>), to provide more elaborate expression matching.</p> <p>processName (<i>required</i>). The name of the process, which is used to identify a specific deployed WS-BPEL process. The process name can also contain wildcards like * (<i>for multiple character matching</i>) and ? (<i>for single character matching</i>), to provide more elaborate expression matching.</p> <p>processInstanceId (<i>required</i>). The identifier of the process instance, which is used to identify a running process instance. The process instance identifier can also contain wildcards like * (<i>for multiple character matching</i>) and ? (<i>for single character matching</i>), to provide more elaborate expression matching.</p> <p>timestamp (<i>required</i>). The time when the process instance was generated. This attribute is used to identify process instances that were triggered after the given specific time in UTC (YYYY-MM-DD hh:mm:ss[.nnnnnnn] [+ -]hh:mm). The time can also contain wildcards like * (<i>for multiple character matching</i>) and ? (<i>for single character matching</i>), to provide more elaborate expression matching.</p>	<p>ResultsArray. An array with the matching process instances. Each process instance entry (see class ProcessInstance in the diagram of Figure 6) in the results array comprises the following five attributes: (i) the process instance identifier (processInstanceId); (ii) the process instances name (processName); (iii) the status of the process instance (status), that can be RUNNING, FINISHED, or FAILED; (iv) its creation time (timestamp) in UTC time; and (v) the identifier of the user (userId) that created the process instance.</p>
<p>listVariables. Allows users to retrieve information about the initialized variables of a specific process instance (i.e. the variables that have been used at least once during the execution of the process instance).</p>	<p>processName (<i>required</i>). As described in the listProcessInstances.</p> <p>processInstanceId (<i>required</i>). As described in the listProcessInstances.</p> <p>scopeId (<i>required</i>). The variable's scope identifier, which is used to identify a specific scope of a process instance variable. The scope identifier can also contain wildcards like * (<i>for multiple character matching</i>) and ? (<i>for single character matching</i>), to provide more elaborate expression matching.</p> <p>scopeName (<i>required</i>). The variable's scope name, which is used to identify the scope name of a process instance variable. The scope name can also contain wildcards like * (<i>for multiple character matching</i>) and ? (<i>for single character matching</i>), to provide more elaborate expression matching.</p>	<p>ResultsArray. An array with the matching results. Each entry in the results array is represented as a ProcessInstanceVariable object (as shown in Figure 6) and has the following six attributes: (i) the process name (processName); (ii) the process instance identifier (processInstanceId); (iii) scope identifier of the variable (scopeId); (iv) the scope name of the variable (scopeName); (v) the variable's name (variableName); and (vi) endpoint address of the P2P Engine Worker node that maintains the variable, in the context of the specified process instance (workerAddress).</p>

<p>deleteProcessInstance. Allows users to remove all monitoring information related to all process instances that match the specified criteria.</p>	<p>processName (<i>required</i>). As described in the listProcessInstances.</p> <p>processInstanceId (<i>required</i>). As described in the listProcessInstances.</p>	<p>Number. Returns the number of process instances the monitoring information of which was removed from the Client's database.</p>
<p>getVariableValue. Allows users to retrieve intermediate results, while a particular instance is being executed.</p>	<p>processName (<i>required</i>). As described in the listProcessInstances.</p> <p>processInstanceId (<i>required</i>). As described in the listProcessInstances.</p> <p>scopeId (<i>required</i>). As described in the listVariables.</p> <p>scopeName (<i>required</i>). As described in the listVariables.</p> <p>variableName (<i>required</i>). The variable's name which is used to identify a the process variable. The variable name can also contain wildcards like * (<i>for multiple character matching</i>) and ? (<i>for single character matching</i>), to provide more elaborate expression matching.</p> <p>workerAddress (<i>required</i>). The endpoint address of the P2P Engine Worker node, that maintains the requested variable. This information allows the Client to directly contact the Worker node in order to retrieve the variable value.</p>	<p>Value. Contains the actual value of the specified variable.</p>

3 Installation Guide

This section is intended to walk the Manager of the ENVISION platform through all the steps necessary for the installation of the second version of the ENVISION Adaptive Execution Infrastructure. This version comprises a number of loosely coupled server-side components, which, as Figure 7 shows, should be deployed on a distributed environment. Then, for each installed components, separate configuration procedures need to be performed, by properly setting their various properties.

Before proceeding with the detailed instructions and configuration procedures, it is worth mentioning that not all components are integrated in the current release of the prototype. In particular, as the UML deployment diagram in Figure 7 shows, the SCS Engine and the Data Mediation Engine are currently provided as standalone tools, and, in this regard, their installation might be considered as optional. These components will be integrated with and used by the Deployment Service and the Service Orchestration Engine in the final release of the ENVISION Adaptive Execution Infrastructure. Nevertheless, for the sake of completeness, this section includes installation and configuration instructions both for the SCS Engine and the Data Mediation Engine.

3.1 Service Orchestration Engine

This section presents the steps required for the installation of the Service Orchestration Engine.

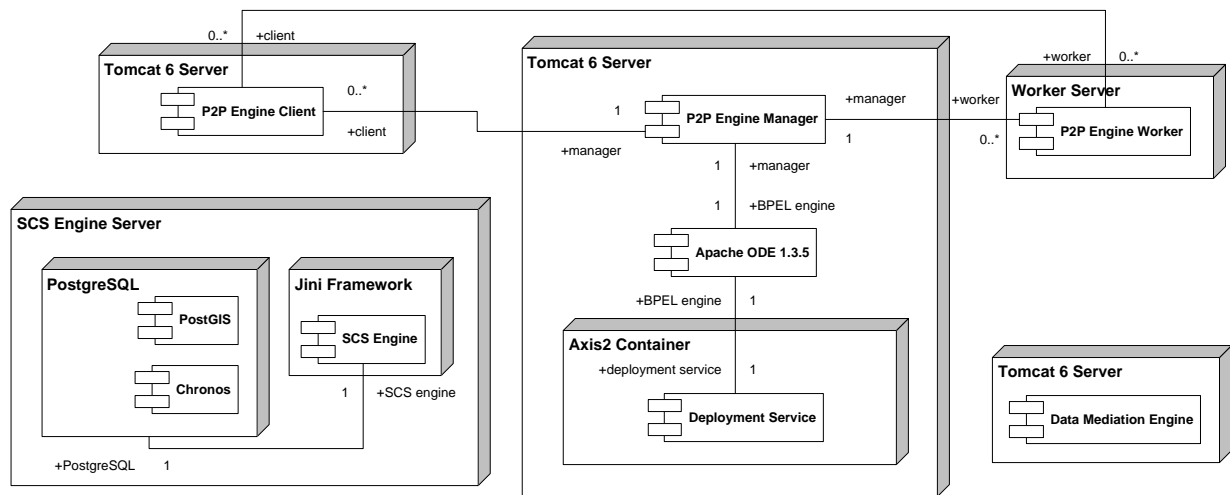


Figure 7: Deployment of the ENVISION Adaptive Execution Infrastructure

3.1.1 Prerequisites

As it is shown in the UML Deployment diagram of Figure 7, the installation of the Service Orchestration Engine requires a distributed environment consisting of: (i) one machine that will host the centralised BPEL engine with the P2P Engine Manager node; (ii) one machine for each P2P Engine Worker node; and (iii) one machine for each P2P Engine Client node. Further, it is required that the **Java Runtime Environment (JRE) v1.6.x** is properly installed to all machines. Details on the available Java downloads can be found in the Java technology's official website⁸. In addition, it is required that **Apache Tomcat 6.x** be installed on the machine that will host the centralised BPEL engine with the P2P Engine Manager node, as well as on all machines that will host Client nodes. Apache Tomcat 6.x can be downloaded from the official website⁹, which also provides detailed instructions for its installation and use.

In the following paragraphs, we will make use of the environment variable **CATALINA_HOME** to denote the home directory of the Tomcat installation. It is also assumed that, the user responsible for the installation of the Service Orchestration Engine is familiar with editing **.properties** files.

3.1.2 Installation

The Service Orchestration Engine is built on top of the **Apache ODE** BPEL engine, and therefore it requires that the ODE 1.3.5 engine is installed on Tomcat. To do so, make sure that Tomcat is stopped and download the binary WAR distribution (**apache-ode-war-1.3.5.zip**) from Apache ODE's official website¹⁰. Next, follow the detailed installation guide for the WAR distribution, which is also available at the official website of the Apache ODE project.

Having installed the Apache ODE 1.3.5 WAR distribution, proceed with downloading the bundle named **SOEngineP2PManager.zip**¹¹ containing all necessary ENVISION extensions to the ODE 1.3.5 BPEL engine. This bundle contains a series of JAR files, which need to be copied in directory

CATALINA_HOME/webapps/ode/WEB-INF/lib

To complete the installation, add the lines shown in Listing 1 to the **web.xml** file of Apache ODE, which is located at the directory **CATALINA_HOME/webapps/ode/WEB-INF**. These lines are required to enable the P2P Engine Servlet, which is used for the configuration of the Worker and Client nodes of the P2P Engine, as we will see in the following paragraphs.

⁸<http://www.oracle.com/technetwork/java/index.html>

⁹<http://tomcat.apache.org>

¹⁰<http://ode.apache.org/>

¹¹<http://kenai.com/projects/envision/downloads/download/Execution%20Infrastructure/M24-Release/SOEngineP2PManager.zip>

```

...
<servlet>
  <servlet-name>P2PEngineServlet</servlet-name>
  <display-name>P2P Engine Servlet</display-name>
  <servlet-class>gr.uoa.di.s3lab.envision.runtime.p2p.P2PEngineServlet</servlet-class>
</servlet>
<servlet-mapping>
  <servlet-name>P2PEngineServlet</servlet-name>
  <url-pattern>/servlet/P2PEngineServlet</url-pattern>
</servlet-mapping>
...

```

Listing 1: Enabling the P2P Engine Servlet in Apache ODE's web.xml

In order to install a Worker node on a machine, download the bundle named **SOEngineP2PWorker.zip**¹². Then, unzip the content of the downloaded bundle into a user-specified directory, which will be referred to as **WORKER_DIR** in the subsequent sections.

Similar to the installation of Worker nodes, to install a Client node on a machine, download the bundle named **SOEngineP2PClient.zip**¹³. To finalize the installation, unzip the bundle, and copy the **MonitorService.war** file in the directory **CATALINA_HOME/webapps** of the machine that will host the Client node.

3.1.3 Configuration

Configuration of the Service Orchestration Engine requires a number of steps for each distinct component, which are separately described in the following paragraphs. Nevertheless, in order to perform the necessary configuration, make sure that the Tomcat server is stopped.

Apache Tomcat and ODE BPEL Engine. To ensure adequate memory capacity upon execution of BPEL processes within the Tomcat container, set the environment variable **CATALINA_OPTS** as follows:

```
CATALINA_OPTS = -Xms1024M -Xmx2048M
```

Further, the execution of some environmental models may be considerable time-consuming, which may lead to unwanted timeouts. To overcome such situations, create a file named **default.endpoint** and save it into folder

```
CATALINA_HOME/webapps/ode/WEB-INF/conf
```

The file is used to set a number of timeout-related properties, and **MUST** include the following lines:

- `mex.timeout=18000000`
- `http.connection.timeout=4000000`
- `http.socket.timeout=40000000`

Of course, the above values are indicative and should be appropriately changed to accommodate specific environment needs. By setting the above values, we increase the time for a connection timeout, allowing the Service Orchestration Engine to cope with Web Services that require longer timing periods for invocation and for receiving their results.

P2P Engine Manager. To configure the P2P Engine Manager node that is embedded into the centralised BPEL engine, create a file named **p2p.properties** in order to set:

- The absolute path to the directory that will be used as home of the Manager¹⁴.

¹²<http://kenai.com/projects/envision/downloads/download/Execution%20Infrastructure/M24-Release/SOEngineP2PWorker.zip>

¹³<http://kenai.com/projects/envision/downloads/download/Execution%20Infrastructure/M24-Release/SOEngineP2PClient.zip>

¹⁴If the installation is performed on a UNIX-based system, make sure that the specified directory is granted write permissions.

- The name of the Manager
- The name of the domain (or, equivalently, peer group) that is initiated by the Manager
- The TCP (Transmission Control Program) port that the Manager will use to listen for incoming connections by Worker and/or Client nodes
- The logging level for the Manager, which determines which messages will be logged upon execution

This file must be saved in folder **CATALINA_HOME/webapps/ode/WEB-INF/classes**. A typical example of the `p2p.properties` file in a Windows-based system is given in Listing 2.

```
home=file:///Users/michael/ENVISION/P2PEngineManager
name=P2PEngineManager
domain=ENVISION
port=9700
log.level=DEBUG
```

Listing 2: Example of the `p2p.properties` file

The P2P Engine Manager will be automatically started together with the Apache ODE engine.

P2P Engine Worker. Each one of the P2P Engine Worker nodes can be configured by properly editing its accompanying `worker.properties` file. More specifically, the following properties need to be set:

- The absolute path to the directory that will be used as home of this Worker¹⁴.
- The name of this Worker
- The name of the domain that is initiated by the Manager node
- The TCP port that this Worker will use to listen for incoming connections by the Manager, Worker, or Client nodes
- The JXTA address of the Manager node that this Worker will use for its registration
- The logging level for this Worker node

To ensure consistency, the domain and JXTA address of the Manager node must be retrieved by means of the P2P Engine Servlet, which resides at the machine where Apache ODE is installed (see previous paragraphs). Open a browser and enter the following address in order to execute the servlet and retrieve the necessary settings:

`http://host:port/ode/servlet/P2PEngineServlet`

In the above address, replace `host` with the name or IP address of the host where the Apache ODE engine is running, and `port` with the port number of the respective Tomcat container. The resulting web page, as shown in Figure 8, provides the domain and JXTA address of the P2P Engine's Manager node.

ENVISION Service Orchestration Engine

P2P Engine

Domain:	ENVISION
Manager Address:	urn:jxta:uuid-6879706572634562A500800000000005AFFA91601864398A5F0DB8B93B34EC904

Figure 8: Web page with information produced by the P2P Engine Servlet

An example of the properties file for an installation of a Worker node on a Windows-based system is given in Listing 3. At this point, the Worker node is ready to run. To do so, simply open a terminal console and execute from the **WORKER_DIR** directory the command `java -jar worker.jar`.

```
home=file:///Users/michael/ENVISION/P2PEngineWorker
name=P2PEngineWorker
domain=ENVISION
port=9800
manager.address=urn:jxta:uuid-6879706572634562A500800000000005AFFA91601864398A5F0DB8B93B34EC904
log.level=DEBUG
```

Listing 3: Example of the worker.properties file

P2P Engine Client. Each one of the P2P Engine Client nodes can be configured by properly editing its accompanying **CATALINA_HOME/webapps/MonitorService/WEB-INF/web.xml** file. The user must set the following initial parameters in the MonitorServlet:

- *Home.* The absolute path to the directory that will be used as home of the Client¹⁴.
- *Name.* The name of the Client.
- *Domain.* The name of the peer group that is initiated by the Manager.
- *Port.* The TCP port that the Manager will use to listen for incoming connections by Worker and/or Client nodes.
- *ManagerAddress.* The JXTA address of the Manager.
- *LogLevel.* The logging level for the Client.
- *IPAddress.* The IP address of the Client's proxy.

As in the case of the Worker nodes, the Domain and ManagerAddress properties of each Client node must be configured with the help of the P2P Engine Servlet. Listing 4 demonstrates a well-configured web.xml file. After completion of the configuration, restart the Apache Tomcat server; the P2P Engine Client will be automatically started as a Web application.

3.2 Deployment Service

This section details the installation and configuration procedures for the Deployment Service.

3.2.1 Prerequisites

The Deployment Service is offered as an Axis2 Web service and thus it requires that Axis2 is installed on the system. To do so, download the binary distribution of **Axis2 1.5.x**, and then follow the detailed installation guide, which is available at the official website of the Apache Axis2 project¹⁵. As it is shown in Figure 7, the Axis2 container is deployed on the same Tomcat server that hosts the Apache ODE BPEL engine. Hence, before proceeding with the installation of the Deployment Service, make sure that the Service Orchestration Engine has been properly installed and configured. In what follows, we assume that the Axis2 distribution is installed as a web application in Tomcat, and make use of the **AXIS2_HOME** environment variable to point at directory

CATALINA_HOME/webapps/axis2

3.2.2 Installation

To install the Deployment Service, simply download the AAR file named **EnvisionDeployService.aar** from the ENVISION project's Downloads section in kenai.com¹⁶. Then, copy the downloaded file to directory **AXIS2_HOME/WEB-INF/services** to complete the installation.

¹⁵<http://axis.apache.org/axis2/java/core/docs/installationguide.html>

¹⁶<http://kenai.com/projects/envision/downloads/download/Execution%20Infrastructure/M24-Release/EnvisionDeployService.aar>

```

<servlet>
  <description>Monitor Servlet for deployed processes in Envsion Engine</description>
  <display-name>MonitorServlet</display-name>
  <servlet-name>MonitorServlet</servlet-name>
  <servlet-class>gr.di.uoa.servlet.MonitorServlet</servlet-class>
  <init-param>
    <description></description>
    <param-name>Home</param-name>
    <param-value>/Users/michael/ENVISION/P2PEngineClient</param-value>
  </init-param>
  <init-param>
    <description></description>
    <param-name>Name</param-name>
    <param-value>Client</param-value>
  </init-param>
  <init-param>
    <description></description>
    <param-name>Domain</param-name>
    <param-value>ENVISION</param-value>
  </init-param>
  <init-param>
    <description></description>
    <param-name>ManagerAddress</param-name>
    <param-value>urn:jxta:uuid-6879706572634562A500800000000005AFFA91601864398A5F0DB8B93B34EC904
  </param-value>
  </init-param>
  <init-param>
    <description></description>
    <param-name>Port</param-name>
    <param-value>9300</param-value>
  </init-param>
  <init-param>
    <description></description>
    <param-name>LogLevel</param-name>
    <param-value>DEBUG</param-value>
  </init-param>
  <init-param>
    <description></description>
    <param-name>IP</param-name>
    <param-value>127.0.0.1</param-value>
  </init-param>
  <load-on-startup>40</load-on-startup>
</servlet>

```

Listing 4: Example of the web.xml file that is used to configure the P2P Engine Client node

3.2.3 Configuration

Before using the Deployment Service, a number of properties need to be properly set. Make sure that the Tomcat server is stopped, and then open the EnvisionDeployService.aar file with an appropriate archive viewer (e.g. WinRAR¹⁷). Extract the **deploy.properties** file, in order to edit it and set the following properties:

- Endpoint address of the Apache ODE deployment service
- Host name of the machine where the Apache ODE BPEL engine is deployed
- Location of the **deployedProcesses.properties** file, which the Deployment Service will use to keep track of the deployed BPEL processes

Listing 5 demonstrates the contents of the deploy.properties file, assuming that the Apache ODE BPEL engine is installed on a Windows-based machine with host name `jupiter.di.uoa.gr`, and that the Tomcat server listens at port 8080.

Note that, the first line in our example refers to property `deployer.impl`, the value of which is fixed and should not be changed by the user. Having set the aforementioned properties, save the `deploy.properties` file and put it back to the `EnvisionDeployService.aar` file, so that the updated file overwrites the original one. At this point, the configuration is complete and the Tomcat server can be started again.

¹⁷<http://rarlab.com/>

```

deployer.impl=gr.uoa.di.s3lab.envision.runtime.services.deploy.util.CentralizedBPELProcessDeployer
apache.ode.deploy.service=http://jupiter.di.uoa.gr:8080/ode/processes/DeploymentService
apache.ode.host.name=jupiter.di.uoa.gr
deployed.process.properties.file=/Users/michael/ENVISION/deployedProcesses.properties
    
```

Listing 5: Example contents of the deploy.properties file

3.3 Semantic Context Space Engine

3.3.1 Prerequisites

In this version, the Semantic Context Space Engine needs to be installed on a Windows-based system, and requires also the installation of a **PostgreSQL** database. Furthermore, to support the manipulation of spatial and temporal data, the database must be equipped with two extensions, namely the **PostGIS** and the **Chronos Temporal Toolkit**. More specifically:

- Version 9.0.6 of the PostgreSQL can be downloaded from the official site of the PostgreSQL¹⁸ and installed according to the given instructions. During the installation, you will be asked to create a database user with specific username and password. Make sure you remember those details, because they will be used in the configuration phase.
- To embed the PostGIS extension to the installed PostgreSQL database, refer to the official site of the former¹⁹.
- To provide the installed PostgreSQL database with the ability to deal with temporal data, the Chronos Temporal Toolkit needs to be downloaded and installed from its official website²⁰ by following the instructions given.

3.3.2 Installation

The Semantic Context Space Engine uses the JavaSpaces²¹ technology of the Jini Framework as its underlying basis. In order to embed the desired semantic and spatiotemporal extensions to the JavaSpaces, we have implemented an extended version of the Jini Framework. This extended Jini offering the required functionality is available as a ZIP file named **SCSEngine.zip**²². As soon as the file is downloaded, simply unzip it to any location in the SCS Engine Server to complete the installation process. In the following, we will use the environment variable **SCS_HOME** to refer to the location where the SCSEngine.zip was unzipped.

3.3.3 Configuration

Before starting the SCS Engine, a number of properties need to be properly set in files **semoutrigger.config** and **startAll.config**, which are both located at:

SCS_HOME/SCSEngine/installverify/support

Open the semoutrigger.config file and set appropriate values to the following properties:

- **installation_dir**: Holds the absolute installation path of the SCS Engine, which takes the form **SCS_HOME/SCSEngine**.

¹⁸<http://www.postgresql.org/>

¹⁹<http://postgis.refrains.net/>

²⁰<http://sourceforge.net/projects/chronosdb/>

²¹<http://java.sun.com/developer/technicalArticles/tools/JavaSpaces/>

²²<http://kenai.com/projects/envision/downloads/download/Execution%20Infrastructure/M24-Release/SCSEngine.zip>

envision

- **server_port**: This is the port of the embedded HTTP server of the SCS Engine. Note that, this server is used to offer the Data Acquisition Interface as a Web service.
- **rdfs_wsml**: This property determines the kind of ontologies that will be supported by the SCS Engine and, in the current version of the component, should be set to either RDFS or WSML.
- **sesame_dir**: This property should be set in case the rdfs_wsml property was set to RDFS, and points at the directory where the internal Sesame Repository²³ will store files required for its proper functioning. It is recommended that the directory specified by the sesame_dir property resides in the **SCS_HOME**.
- **ontology_name**: Holds the name of the ontology that will be used by the SCS Engine. In the final version of the component, the use of multiple ontologies will be enabled and thus this property will no longer be needed.
- **postgresql_url**: This is the URL of the PostgreSQL database that is used by the SCS Engine.
- **postgresql_username**: This is the username required for connecting to the PostgreSQL database.
- **postgresql_password**: This is the password required for connecting to the PostgreSQL database.

```
/** JiniDir and ServerPort plugin configuration settings
**/
com.sun.jini.outrigger.semanticspaceserver.jini_server {

    installation_dir = "C:\\SCSEngine\\";
    server_port = "8088";

}

/** WSML RDFS plugin configuration settings
**/
com.sun.jini.outrigger.semanticspaceserver.wsml {

    //for wsml
    //rdfs_wsml = "WSML";
    //ontology_name = "C:\\SCSEngine\\Ontologies\\amazonOntology.wsml";

    //for rdfs
    rdfs_wsml = "RDFS";
    sesame_dir = "C:\\SCSEngine\\Ontologies\\myRepository\\";
    ontology_name = "C:\\SCSEngine\\Ontologies\\SwingDomainOntology.xml"

}

/** Postgres plugin configuration settings
**/
com.sun.jini.outrigger.semanticspaceserver.postgresql {

    postgresql_url = "jdbc:postgresql://pleiades.di.uoa.gr/postgis";
    postgresql_username = "postgres";
    postgresql_password = "12345678";

}
}
```

Listing 6: Example of the semoutrigger.config file

Next, open the startAll.config file and set the **installation_dir** property so that it points to

SCS_HOME/SCSEngine

```
private static installation_dir="C:\\SCSEngine";
```

Listing 7: Example of the startAll.config file

Listings 6 and 7 provide excerpt examples of the two files, where all the aforementioned properties are set.

²³<http://www.openrdf.org/>

After applying the above described settings, the SCS Engine can be started by executing the **startSCSEngine.bat** file, which is located at **SCS_HOME/SCSEngine**. As soon as the SCS Engine is started, the Data Acquisition Interface can be accessed through the following WSDL URL:

```
http://hostname:port/jetty_axis/services/SemanticJavaSpace?wsdl
```

In the above URL, simply replace the *hostname* and *port* placeholders with their server-specific values.

3.4 Data Mediation Engine

3.4.1 Prerequisites

The prerequisites for installing the Data Mediation Engine are minimal. What is needed is a running Apache Tomcat version 6.x. The service contains all needed dependencies (e.g. libraries), hence no modification is needed on the Apache Tomcat side.

3.4.2 Installation

Installing the Data Mediation Engine is straight forward. One needs to copy the .war file²⁴ to the **webapps** directory that is used by the running Apache Tomcat instance.

3.4.3 Configuration

The Data Mediation Engine runs out-of-the-box, hence no particular configuration steps need to be followed.

4 User Guide

In this section, we present a series of typical scenarios to demonstrate how the ENVISION Adaptive Execution Infrastructure is accessed and used. Without loss of generality, all presented scenarios revolve around the deployment, execution, monitoring, and undeployment of a simple demo process. The respective use case²⁵ was originally specified and designed by BRGM, while contributions were made by all technical partners of the ENVISION consortium.

Figure 9 provides a *BPMN (Business Process Model and Notation)* [12] diagram of the demo process, which implements a simplified drought risk assessment model. The goal of that model is to help scientists in assessing the potential risk of droughts for different areas of France. To achieve this goal, the model invokes three *OGC (Open Geospatial Consortium)*²⁶ services according to the control flow depicted in the BPMN diagram. The functionality of each one of these services is summarized as follows:

- The **Sensor Observation Service (SOS)** [9] retrieves French Ground Water level observations using sensors, which provide piezometer measurements.
- The **Web Feature Service (WFS)** [10] provides access to a feature set of urbanised areas. The data comes from *CORINE (Coordination of Information on the Environment)* [6], a program sponsored by the European Environment Agency²⁷ to gather environmental information through the classification of remote sensing data.

²⁴The Data Mediation Engine is available for download at <http://sesa.sti2.at/envision/>

²⁵A full description of the demo process is available online at <http://envision.brgm-rec.fr/SimpleUseCase.aspx>

²⁶<http://www.opengeospatial.org/>

²⁷<http://www.eea.europa.eu/>

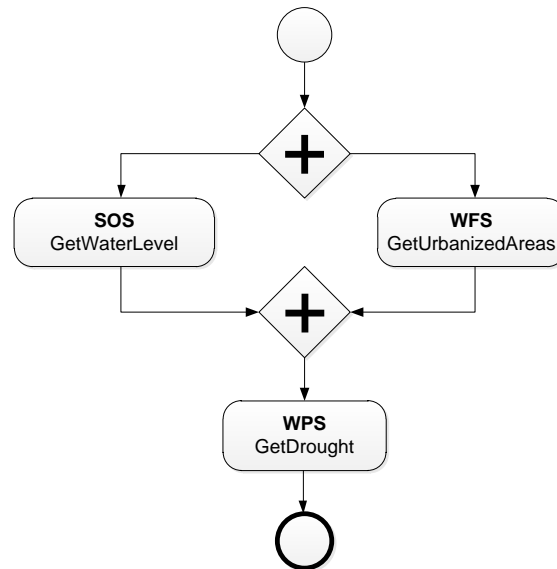


Figure 9: The drought risk assessment model

- The **Web Processing Service (WPS)** [11] implements an algorithm that calculates which areas are in potential risk of drought in France. The underlying idea is based on setting a threshold for groundwater level, considering that levels below this threshold indicate risk of drought, and find out which areas present such a risk. The service takes as input a feature collection from the previously described WFS, and an observation collection from the above mentioned SOS.

4.1 Target Users

The ENVISION Adaptive Execution Infrastructure is intended to be used by actors playing two different roles, namely the **Designer** role, and the **End-user** role. These roles have been defined in ENVISION deliverable D2.1 [D2.1], and are used hereinafter to identify the kind of user that interacts with the execution infrastructure at each one of the scenarios.

4.2 Deployment of the Drought Risk Assessment Model

Assuming that the development of the drought risk assessment model has finished, and that all related files have been generated, the Designer makes use of the Resource Portlet in order to deploy the model to the execution infrastructure. All selected files are packaged into a zip bundle, which is subsequently submitted for deployment by accessing and invoking the Deployment Service through the Deployment Interface. In the case of successful deployment, the URL of the WSDL corresponding to the drought risk assessment WS-BPEL process is returned to the Designer. This URL is necessary not only to enable the execution of the landslide WS-BPEL process, but also to allow the Designer to undeploy it in the future, if it is required. Listings 8 and 9 show the SOAP request and response messages, respectively, for the invocation of the deploy operation.

4.3 Execution of the Drought Risk Assessment Model

The End-user of the drought risk assessment model (e.g. a scientist working at BRGM) is able to execute it by making use of the Execution Portlet. The latter provides a user-friendly interface to assist the End-user in preparing the initial input and submitting it by invoking the drought risk assessment WS-BPEL process as a SOAP Web service. Listing 10 displays an example of the SOAP request containing the initial input to the WS-BPEL process.

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:dep="http://s3lab.di.uoa.gr/envision/runtime/services/deploy">
  <soapenv:Header/>
  <soapenv:Body>
    <dep:DeployRequestParameters>
      <dep:zipBundle>{zip-bundle-as-byte-array}</dep:zipBundle>
      <dep:processName>DroughtRiskAssessmentProcess</dep:processName>
    </dep:DeployRequestParameters>
  </soapenv:Body>
</soapenv:Envelope>
```

Listing 8: SOAP request for the deploy operation

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:dep="http://s3lab.di.uoa.gr/envision/runtime/services/deploy">
  <soapenv:Header/>
  <soapenv:Body>
    <dep:DeployResponseParameters>
      <dep:deployed>true</dep:deployed>
      <dep:wSDLURL>
        http://jupiter.di.uoa.gr:8080/ode/processes/DroughtRiskAssessmentProcessPort?wSDL
      </dep:wSDLURL>
      <dep:deployDetails>
        DroughtRiskAssessmentProcess-1: {urn:envision:bpel}DroughtRiskAssessmentProcess-1
      </dep:deployDetails>
    </dep:DeployResponseParameters>
  </soapenv:Body>
</soapenv:Envelope>
```

Listing 9: SOAP response for the deploy operation

Before submitting the SOAP request to the WS-BPEL engine, the Execution Portlet appends the End-user's ID to the header part of the message. This is required to enable the association of the generated process instance with that particular user, so that the user is able to retrieve monitoring information while the process instance is being executed. Indeed, as it is shown in Figure 10, the WS-BPEL engine extracts the User ID from the submitted SOAP request, and employs the Manager node in order to send it along with the generated process instance ID to the Client node representing the End-user²⁸. This pair of IDs is stored to the Client node's embedded database along with additional metadata regarding the process instance, such as its creation timestamp, name, status, etc. This information is made available to the End-user through the Monitoring Portlet, and can be used for the retrieval of intermediate results while the process instance is being executed.

In the meantime, the process instance is executed, and the Manager employs the Worker nodes in order to distribute the resource-demanding activities involving interactions with external services and complex data mappings. The Worker nodes are also responsible for the maintenance of the variables that are used by the process instance to hold the produced/consumed data. Thus, to enable the retrieval of intermediate results, the Manager sends to the Client node the endpoint addresses of the Workers in charge of the process's variables. Upon completion of the process instance execution, the End-user will receive the final results in the form of a SOAP response, which is sent from the WS-BPEL engine back to the Execution Portlet. An excerpt of such a response, which is the same as the output of the WPS service in our example, is shown in the Listing 11.

4.4 Monitoring of the Drought Risk Assessment Model

While the drought risk assessment WS-BPEL process is being executed, the End-user can use the monitoring mechanism provided by the ENVISION Adaptive Execution Infrastructure, in order to keep track of the process instance, and retrieve intermediate results. Through the Monitoring Portlet, the End-user may invoke the various Web service operations that are offered by the Monitoring Interface of the Client node.

As it is shown in Figure 11, in order to retrieve the ID of the particular process instance, the End-user

²⁸The same Client node may represent more than one End-users.

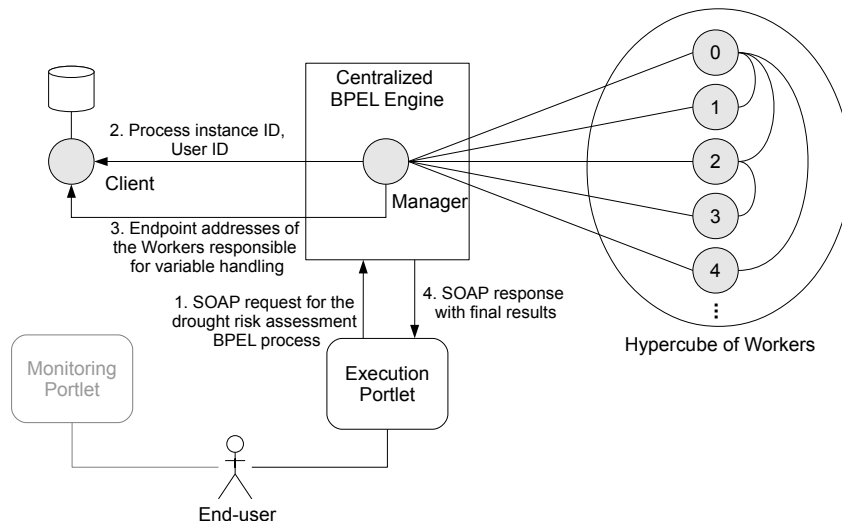


Figure 10: Execution of the drought risk assessment model

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:ns="http://www.opengis.net/wps/1.0.0"
  xmlns:envision="http://s3lab.di.uoa.gr/envision">
  <soapenv:Header>
    <envision:USER-ID>...</envision:USER-ID>
  </soapenv:Header>
  <soapenv:Body>
    <ns:Execute/>
  </soapenv:Body>
</soapenv:Envelope>
```

Listing 10: SOAP request for the execution of the Drought Risk Assessment Model

invokes the listProcessInstances operation by formulating and submitting the appropriate SOAP request to the Client. An example of such request is illustrated in Listing 12. In turn, the Client executes the request and returns to the Monitoring Portlet the list of process instances that belong to that specific End-user. An example of the corresponding SOAP response is given in Listing 13.

The list of process instances is displayed in the Monitoring Portlet, and the End-user is able to select the process instance she is interested in monitoring. As soon as the process instance is selected, the End-user may view its variables by invoking the listVariables Web service operation of the Monitoring Interface. The SOAP request of such an invocation is exemplified in Listing 14. The Client responds to that request by returning information about all variables that have been initialized in the context of the selected process instance. As Listing 15 shows, among the returned information is the address of the Worker node in charge of each variable, which can be used by the Client to directly retrieve the value of a selected variable.

For instance, as it is shown in Figure 12, the End-user may choose to view the value of the SOS output variable, while the drought risk assessment model is still being executed. In doing so, the End-user invokes the GetVariableValue Web service operation on the Client node. The latter resolves the Worker in charge of the SOS output variable for the specified process instance (in our example, let us assume this is Worker node with Hypercube ID 2) and sends a peer-to-peer request to it in order to retrieve the variable value. The Worker node sends a peer-to-peer response with the requested variable value to the Client, which forwards it to the Monitoring Portlet via a SOAP response²⁹.

²⁹It should be noted that, intermediate variable values are only maintained by the responsible Worker nodes for as long as the respective process instance is being executed. The Client node's database only keeps meta-data about the process instances and their variables.

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:ns="http://www.opengis.net/wps/1.0.0" service="WPS" version="1.0.0"
  xmlns:ns1="http://www.opengis.net/ows/1.1"
  xmlns:gml="http://www.opengis.net/gml">
  <soapenv:Header/>
  <soapenv:Body>
    <ns:ExecuteResponse ...>
      <ns:Process ns:processVersion="1.0.0">
        <ns1:Identifier>org.n52.wps.server.algorithm.intersection.IntersectionAlgorithm</ns1:Identifier>
        <ns1:Title>org.n52.wps.server.algorithm.intersection.IntersectionAlgorithm</ns1:Title>
      </ns:Process>
      <ns:Status creationTime="2012-01-05T21:29:01.381+01:00">
        <ns:ProcessSucceeded>The service successfully processed the request.</ns:ProcessSucceeded>
      </ns:Status>
      <ns:ProcessOutputs>
        <ns:Output>
          <ns1:Identifier>intersection_result</ns1:Identifier>
          <ns1:Title>intersection_result</ns1:Title>
          <ns:Data>
            <ns:ComplexData encoding="UTF-8" mimeType="text/xml">
              <gml:FeatureCollection ...>
                ...
              </gml:FeatureCollection>
            </ns:ComplexData>
          </ns:Data>
        </ns:Output>
      </ns:ProcessOutputs>
    </ns:ExecuteResponse>
  </soapenv:Body>
</soapenv:Envelope>
```

Listing 11: SOAP response from the execution of the Drought Risk Assessment Model

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:mon="http://monitor.runtime.envision.s3lab.di.uoa.gr">
  <soapenv:Header/>
  <soapenv:Body>
    <mon>ListProcessInstancesRequest>
      <mon:userId>jpogas</mon:userId>
      <mon:processName>*</mon:processName>
      <mon:processInstanceId>*</mon:processInstanceId>
      <mon:timestamp>*</mon:timestamp>
    </mon>ListProcessInstancesRequest>
  </soapenv:Body>
</soapenv:Envelope>
```

Listing 12: SOAP request for the listProcessInstances operation

4.5 Undeploying the Drought Risk Assessment Model

At any time, the Designer of the drought risk assessment model may undeploy it from the ENVISION Adaptive Execution Infrastructure by invoking the Undeploy operation of the Deployment Service. Through the Resource Portlet, the Designer is able to locate the URL of the WSDL document that corresponds to the deployed WS-BPEL process, and send it via the Deployment Interface as input for the undeployment. The respective SOAP request and response messages for the undeployment of the drought risk assessment model in our example are given in Listings 16 and 17.

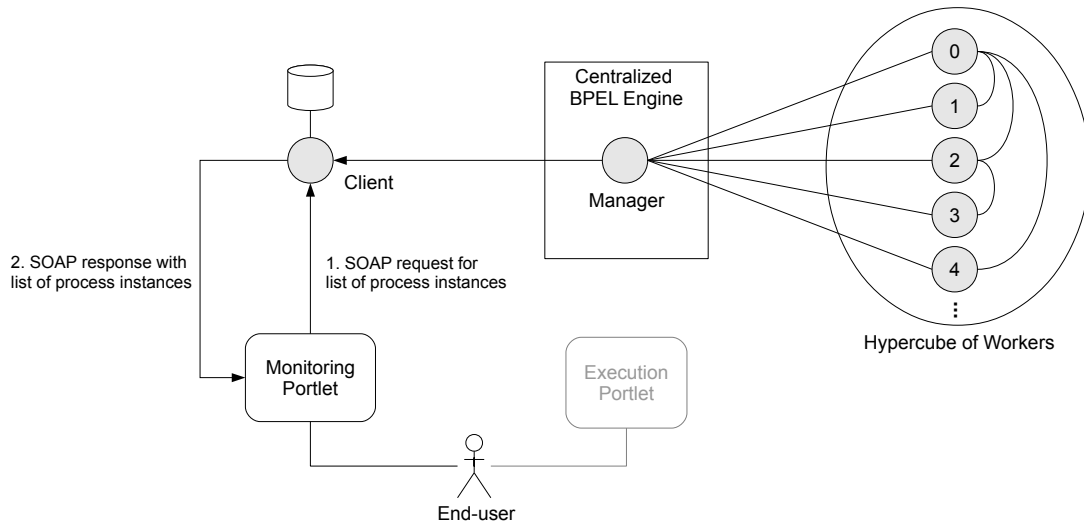


Figure 11: Retrieving the list of process instances through the Monitoring Interface

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/">
  <soapenv:Body>
    <mon:ListProcessInstancesResponse
      xmlns:mon="http://monitor.runtime.envision.s3lab.di.uoa.gr">
      <mon:processInstanceId>1</mon:processInstanceId>
      <mon:processName>{urn:envision:bpel}DroughtRiskAssessmentProcess-1</mon:processName>
      <mon:status>RUNNING</mon:status>
      <mon:timestamp>18:00:00</mon:timestamp>
      <mon:userId>jpogas</mon:userId>
    </mon:ListProcessInstancesResponse>
  </soapenv:Body>
</soapenv:Envelope>
```

Listing 13: SOAP response of the listProcessInstances operation

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:mon="http://monitor.runtime.envision.s3lab.di.uoa.gr">
  <soapenv:Header/>
  <soapenv:Body>
    <mon:ListVariablesRequest>
      <mon:processName>{urn:envision:bpel}DroughtRiskAssessmentProcess-1</mon:processName>
      <mon:processInstanceId>1</mon:processInstanceId>
      <mon:scopeId>*</mon:scopeId>
      <mon:scopeName>*</mon:scopeName>
    </mon:ListVariablesRequest>
  </soapenv:Body>
</soapenv:Envelope>
```

Listing 14: SOAP request for the listVariables operation

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/">
  <soapenv:Body>
    <mon:ListVariablesResponse xmlns:mon="http://monitor.runtime.envision.s3lab.di.uoa.gr">
      <mon:processVariables>
        <mon:variable>
          <mon:processName>{urn:envision:bpel}DroughtRiskAssessmentProcess-1</mon:processName>
          <mon:processInstanceId>1</mon:processInstanceId>
          <mon:scopeId>3</mon:scopeId>
          <mon:scopeName>_PROCESS_SCOPE:DroughtRiskAssessmentProcess</mon:scopeName>
          <mon:variableName>DroughtRiskAssessmentProcessOperationIn</mon:variableName>
          <mon:workerAddress>urn:jxta:uuid-687970639...</mon:workerAddress>
        </mon:variable>
        ...
      </mon:processVariables>
    </mon:ListVariablesResponse>
  </soapenv:Body>
</soapenv:Envelope>
```

Listing 15: SOAP response of the listVariables operation

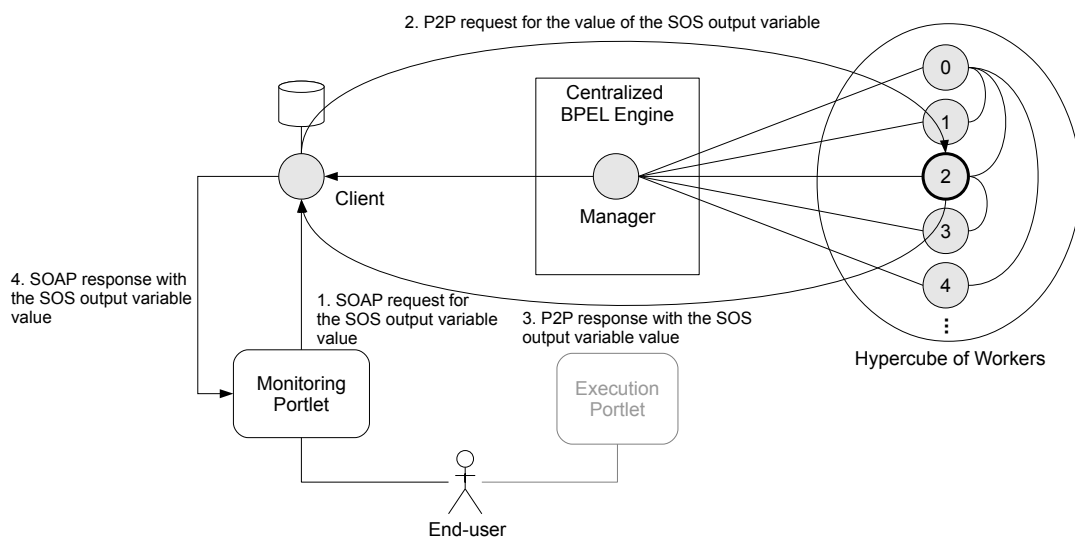


Figure 12: Retrieving the value of the SOS output variable through the Monitoring Interface

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:dep="http://s3lab.di.uoa.gr/envision/runtime/services/deploy">
  <soapenv:Header/>
  <soapenv:Body>
    <dep:UndeployRequestParameters>
      <dep:wSDLURL>
        http://jupiter.di.uoa.gr:8080/ode/processes/DroughtRiskAssessmentProcessPort?wSDL
      </dep:wSDLURL>
    </dep:UndeployRequestParameters>
  </soapenv:Body>
</soapenv:Envelope>
```

Listing 16: SOAP request for the undeploy operation

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/">
  <soapenv:Body>
    <dep:UndeployResponseParameters xmlns:dep="http://s3lab.di.uoa.gr/envision/runtime/services/deploy">
      <dep:undeployed>true</dep:undeployed>
      <dep:undeployDetails/>
    </dep:UndeployResponseParameters>
  </soapenv:Body>
</soapenv:Envelope>
```

Listing 17: SOAP response for the undeploy operation

5 Conclusions

This deliverable presents the second version of the ENVISION Adaptive Execution Infrastructure, and provides details for the installation of its components. As it was mentioned in the respective sections, not all components are currently integrated. Nevertheless, a user guide was also given to demonstrate the usage of the Deployment Service and the Service Orchestration Engine in deploying, executing, monitoring, and undeploying environmental models that are implemented as WS-BPEL processes. Finally, an appendix is included reporting the theoretical work that was carried out during the second year of the project and refers to the ENVISION approach towards data-driven adaptation of service-oriented processes.

The work that will be carried out during the last year of the ENVISION project in Work Package 6 will revolve around and focus on the following axes:

- Completion of the development and integration of all components of the ENVISION Adaptive Execution Infrastructure
- Extensive testing and validation of the produced prototypes against the requirements and use cases of the ENVISION user partners
- Evaluation of the performance of the distributed execution and monitoring of the environmental models as WS-BPEL processes

The results of the above tasks will be documented in ENVISION Deliverable D6.4, which is due in Month 34 of the project. Hence, this document will be expanded and updated as needed in order to provide all details regarding the implementation and evaluation of the final release of the ENVISION Adaptive Execution Infrastructure.

References to ENVISION Deliverables

- [D2.1] J. Langlois, F. Tertre, F. Husson, and P. Maué, ENVISION Deliverable 2.1: Environmental Semantic Web Portal – Architecture specification
<http://www.envision-project.eu/wp-content/uploads/2011/06/D2.1-v1.1.pdf>
- [D2.3] F. Husson, F. Tertre, R. Grønmo, I. Larizgoitia, P. Maué, H. Michels, and M. Grčar, ENVISION Deliverable D2.3: Environmental Semantic Web Portal Version 2 – Improved development and user guide
- [D5.4] I. Larizgoitia and I. Toma, ENVISION Deliverable D5.4: Integration of the open source catalogue with the semantic discovery engine – Version 2
- [D6.1] A. Tsalgatidou, G. Athanasopoulos, Y. Pogas, P. Kouki, and M. Pantazoglou, ENVISION Deliverable D6.1: ENVISION Adaptive Execution Infrastructure – Architecture Specification
<http://www.envision-project.eu/wp-content/uploads/2010/01/D6.1-FINAL.pdf>
- [D6.2] A. Tsalgatidou, G. Athanasopoulos, I. Pogkas, and P. Kouki, ENVISION Deliverable D6.2: Adaptive Execution Infrastructure Version 1 – User Guide
http://www.envision-project.eu/wp-content/uploads/2010/01/D6.2_1.0.pdf

References

- [1] Alexandre Alves, Assaf Arkin, Sid Askary, Charlton Barreto, Ben Bloch, Francisco Curbera, Mark Ford, Yaron Goland, Alejandro Guízar, Neelakantan Kartha, Canyang Kevin Liu, Rania Khalaf, Dieter König, Mike Marin, Vinkesh Mehta, Satish Thatte, Danny van der Rijn, Prasad Yendluri, and Alex Yiu. Web Services Business Process Execution Language Version 2.0. <http://docs.oasis-open.org/wsbpel/2.0/OS/wsbpel-v2.0-OS.html>, 11 April 2007.
- [2] George Athanasopoulos and Aphrodite Tsalgatidou. An approach to data-driven adaptable service processes. In José A. Moinhos Cordeiro, Maria Virvou, and Boris Shishkov, editors, *ICSOFT (1)*, pages 139–145. SciTePress, 2010.
- [3] Piergiorgio Bertoli, Alessandro Cimatti, Marco Roveri, and Paolo Traverso. Planning in nondeterministic domains under partial observability via symbolic model checking. In Bernhard Nebel, editor, *IJCAI*, pages 473–478. Morgan Kaufmann, 2001.
- [4] Eric Christensen, Francisco Curbera, Greg Meredith, and Sanjiva Weerawarana. Web Service Description Language (WSDL) 1.1. <http://www.w3.org/TR/wsdl>, March 2001. W3C Note, World Wide Web Consortium (W3C).
- [5] Jos de Bruijn, Dieter Fensel, Mick Kerrigan, Uwe Keller, Holger Lausen, and James Scicluna. *Modeling Semantic Web Services: The Web Service Modeling Language*. Springer, Berlin, Heidelberg, 2008.
- [6] European Commission.. Directorate General. Environment, Office for Official Publications of the European Communities, Y. Heymann, Nuclear Safety Commission of the European Communities. Directorate-General for Environment, and Civil Protection. *CORINE land cover: technical guide*. Environment and quality of life series. Office for Official Publications of the European Communities, 1994.
- [7] Li Gong. JXTA: A network programming environment. *IEEE Internet Computing*, 5(3):88–95, 2001.
- [8] N. Mitra. SOAP version 1.2 part 0: Primer (second edition). <http://www.w3.org/TR/soap12-part0/>, April 2007. W3C Recommendation.
- [9] Open Geospatial Consortium (OGC). OpenGIS® Sensor Observation Service. OGC® Standard, <http://www.opengeospatial.org/standards/sos>.
- [10] Open Geospatial Consortium (OGC). OpenGIS® Web Feature Service. OGC® Standard, <http://www.opengeospatial.org/standards/wfs>.
- [11] Open Geospatial Consortium (OGC). OpenGIS® Web Processing Service. OGC® Standard, <http://www.opengeospatial.org/standards/wps>.
- [12] Object Management Group (OMG). Business Process Model And Notation (BPMN) Version 2.0. <http://www.omg.org/spec/BPMN/2.0/>, January 2011.
- [13] Marco Pistore, Fabio Barbon, Piergiorgio Bertoli, Dmitry Shaparau, and Paolo Traverso. Planning and monitoring web service composition. In Christoph Bussler and Dieter Fensel, editors, *AIMSA*, volume 3192 of *Lecture Notes in Computer Science*, pages 106–115. Springer, 2004.
- [14] Stuart J. Russell and Peter Norvig. *Artificial Intelligence: a modern approach*. Prentice Hall, 2nd international edition edition, 2003.
- [15] Mario T. Schlosser, Michael Sintek, Stefan Decker, and Wolfgang Nejdl. HyperCuP - Hypercubes, Ontologies, and Efficient Search on Peer-to-Peer Networks. In Gianluca Moro and Manolis Koubarakis, editors, *AP2PC*, volume 2530 of *Lecture Notes in Computer Science*, pages 112–124. Springer, 2002.
- [16] Aphrodite Tsalgatidou, George Athanasopoulos, and Michael Pantazoglou. Interoperability among heterogeneous services: The case of integration of p2p services with web services. *International Journal of Web Service Research*, 5(4):79–110, 2008.
- [17] World Wide Web Consortium (W3C). XML Schema, <http://www.w3.org/XML/Schema>.

envision

[18] World Wide Web Consortium (W3C). XSL Transformations (XSLT) Version 1.0. W3C Recommendation, <http://www.w3.org/TR/xslt>, 16 November 1999.

Appendix A Data-driven Adaptation of Environmental Models

A.1 Introduction

The provision of adaptable service-oriented processes is a vision pursued in many research projects using several approaches, e.g. ALLOW³⁰, DiVA³¹, SERCIS³². The goal of such processes, similarly to what applies in other software systems, is to “use available information about changes in its environment to improve its behavior”³³. The approach applied in the ENVISION project is a *data-driven* one [2]. Data-driven process adaptation accommodates the provision of adaptable service processes by exploiting the use of information available to the process environment, in addition to existing services; information here refers to semantically annotated, structured data that is related to the process at hand. Adaptation in the context of our approach includes the use of possible alternatives for the achievement of the goals and sub-goals defined in a process; alternatives include the utilization of available related information and/or services.

The input required by the ensued approach includes a model of the originally specified process in the form of a BPEL description, and semantically enhanced service descriptions (e.g. in the form of WSMO or WSMO-Lite service descriptions). Despite that in the context of Envision the specified process comprises OGC compliant services, e.g. WFS, SOS, WPS, etc., additional types of services, e.g. W3C compliant or Peer-to-Peer services, can be seamlessly used.

In the following, we provide details on the background theoretical model as well as on the process used for the identification of related information elements and alternative process steps.

A.2 Process Adaptation

As it has been articulated in the ENVISION deliverable D6.1, the representation used for the data-driven process adaptation is that of a non-deterministic, partially observable planning problem. Solutions to such problems are in the form of conditional plans; conditional plans contain branching control structures, i.e. if-then-else structures that decide which path will be followed based on the value of a condition.

Definition A.1. A formal representation of the problem domain (D) is a tuple, (S, A, R, O, X) , where:

- S is the finite set of states of the associated state transition system
- A is the finite set of actions $A = \{a_i : i \leq l \wedge l \in \mathbb{N}\}$
- $R \subseteq S \times A \times S$ is the transition relation
- O is a finite state of observation variables $O = \{o_i : i \leq n \wedge n \in \mathbb{N}\}$
- $X : S \times \{\top, \perp\}$ is the relation for the evaluation of observation variables $o \in O$ on each state

In the frame of such planning problems, the transition relation R can map the execution of an action $a \in A$, on a state $s \in S$ (assuming that a is applicable on s) to more than one successor states, i.e. $S' \subseteq S$, $|S'| \geq 1$. An action a is applicable on a state $s \in S$, iff there exists a state $s' \in S$ such that $R(s, a, s')$ stands. The set O contains the finite set of observation variables o_i whose values are evaluated at runtime. The value of each observation variable at each state is defined by the observation relation X . A simplification normally introduced to avoid the unnecessary complexities is to consider observation variables as boolean variables whose values could be either true or false, i.e. \top or \perp . Therefore if $X_o(s, \top)$ holds at a state $s \in S$ then the value of variable o at state s is true. The dual holds in cases where variable o is false. In cases where both $X_o(s, \top)$ and $X_o(s, \perp)$ hold, variable o has an undefined value.

³⁰ALLOW EU FP7 project, <http://www.allow-project.eu/>

³¹DiVA EU FP7 project, <http://www.ict-diva.eu/DiVA>

³²SERCIS EU FP7 project, <http://www.sercis.eu/>

³³Peter@Norvig.com. <http://www.norvig.com/adapaper-pcai.html>

envision

The extensions introduced in this problem specification so as to accommodate the data-driven aspect of our approach that is applied in ENVISION include: a) an observation and action set 'extension' mechanism that facilitates the inclusion of additional information elements and related services, b) a mechanism facilitating the valuation of observations (i.e. observation variables) based on queries executed over an open set of information elements.

Regarding observation valuation, we can safely assume that an observation variable (o_n) of the planning domain D is defined over a finite set of information elements OD_n (see [3]). The assessment of an observation variable at runtime should always return a value within the specified set OD_n . In the frame of our approach the valuation of an observation variable (o_n) is mapped to the assessment of a query (q_n) performed over an information source, which corresponds to the system environment. However, due to its open nature, such an information source may comprise information elements that are irrelevant to an executing process. Thus, the set of queries ($Q = \{q_1, q_2, \dots, q_n\}$) performed over a source should be properly structured so as to avoid the retrieval of erroneous information elements. To accommodate this concern, we provide a semantic-based information discovery mechanism. Queries (q_i) are extracted out of the semantic and syntactic details of the associated observation variables (o_i). The Semantic Context Space (SCS) Engine executes these queries, and matching results are returned back to the running process.

The specification of the observation variables, i.e. their syntactic and semantic properties, is therefore a crucial aspect of our approach. In the following, we provide details on the observation identification and expansion mechanism. In addition to the observation variables (O) this mechanism facilitates the expansion of the actions set (A) by introducing additional services that can exploit the introduced observations.

A.3 Observation Variable Identification

A crucial step on the applied approach is the selection of the candidate observation variables. Nevertheless, given a process specification there are several artifacts that one could possibly observe, e.g. internal variables, interacting services, internal actions, etc. Thus, the selection of candidate observations that maximize the coverage over the possible process states without overwhelming the problem specification with observation variables (Definition A.1) is an issue of paramount importance.

Among the set of candidates, one type of artifacts that has received attention in several approaches is the list of interacting services. Similar to such approaches, we consider external services as the most appropriate artifact for observation. The outcomes of external services constitute a major aspect for the specification of the process execution flow, hence its state.

As in [13], interactions between a composite service (i.e. process) and its external service providers can be modeled via the exchange of messages over a set of communication channels. Read and write operations over the supported communication channels are the means for controlling the evolution of the process and the business protocols of the external service providers (see Figure 13).

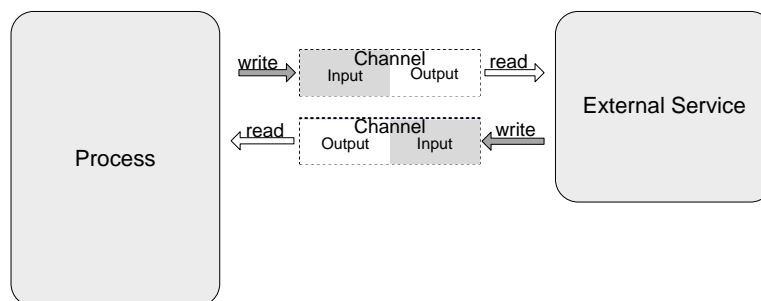


Figure 13: Communication channel between a process and external service providers

For monitoring these interactions, hence the process evolution, one could observe the status of a channel or the variables used for conveying the exchanged messages. In the context of our approach, we have decided to monitor the variables holding the messages intended for the process, i.e. the process is the intended recipient. Even though this selection doesn't provide the maximum coverage over the set

of possible process states, i.e. S in Definition A.1, the resulting observations can adequately capture the state of the process and can be directly associated to information collected from other sources, i.e. other communication channels. Moreover, the introduced number of observations is proportional to the number of interacting services, i.e. proportional with respect to the total number of actions but less in absolute number $|O| < |A|$.

This set of observations constitutes the input used by the expansion mechanism which extends it with the inclusion of related observations and actions. Although additional observations, e.g. observations on the status of the communication channels, could also be used, the set of observations that is fed to the expansion mechanism should only include observations on the variables holding the messages intended for the process.

A.4 Observation and Action Expansion

Our approach towards the use of similar information elements is based on the expansion of observations, i.e. with similar observations, for a given process in a manner controlled by a utilized ontology. Metaphorically, one may say that this approach is like introducing additional sensors that can monitor the process environment. For this extension to be useful, appropriate services (actions) should be also added so as to leverage the use of the introduced observations. Therefore the expansion process can be graphically illustrated as in Figure 14.

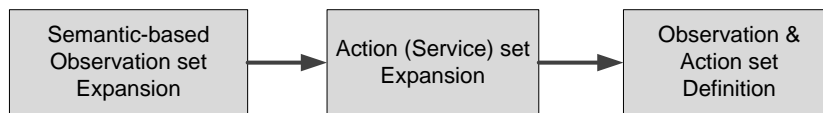


Figure 14: Observation and Action Expansion Process

The expansion process comprises three main steps that facilitate the observation and action expansion. Starting with the '*Semantic-based Observation set Expansion*' step, a set of concepts relevant to the given observations is calculated out of a given ontology. This set is fed to the '*Action set Expansion*' step that returns a set of related services. The final step includes the specification of the extended observations and action sets that will be utilized by the incorporated planner for the calculation of the corresponding condition plan.

These steps are further analyzed in the following.

A.4.1 Observation Concept Expansion

The first step of the expansion process involves the semantic-based extension of the observation set. The underpinning assumption which drives the observation expansion is that instead of considering partially matching results to the performed observations we may as well look for exact matches to *partially matching* (i.e. similar) observations. Hence, the set of observations O linked to a process P is expanded with the introduction of similar observations.

Definition A.2. An observation variable $o \in O$ is defined as a tuple (O_{name}, vc, t) , where:

- O_{name} is the name of the variable
- vc is the semantic concept associated to the given variable
- t is the variable's syntactic type, i.e. it specifies the related domain of values for variable o_i

Each semantic concept associated to the specified observation variables is part of a vocabulary $vc \in V$, which constitutes the process's ontology $(V = \{vc_i : vc_i \text{ concept of ontology}, i \leq n \wedge n \in \mathbb{N}\})$.

To facilitate the expansion process we introduce two additional features i.e. an expansion function ($oExp$) and an expansion ratio (sD). The sD property dictates the minimum similarity distance (i.e. the floor value) between the concepts of an ontology so as to consider them part of the same set (i.e. expansion

set). Thus, given an ontology (V), a concept (vc) and an expansion ratio (sD), the $oExp$ function returns all concepts of the provided ontology, whose similarity to vc is equal to or greater than sD .

$$oExp : (R \times V) \mapsto V \quad (1)$$

For the calculation of the expansion function several concept similarity distance measurements can be used. Such a measure providing an indication of similarity between two concepts based on the IS-A relation hierarchy is Dice's coefficient (S_{Dice}):

$$S_{Dice} = \frac{2 \cdot f(X \cap Y)}{f(X) + f(Y)} \quad (2)$$

With function f reflecting the cardinality of a given set, $f(X \cap Y)$ represents the cardinality of the set from the root to the most specific common ancestor of two given concepts i.e. x and y . Along the same lines, $f(X)$ and $f(Y)$ are the cardinalities of the set of ancestors for x and y respectively.

We need to state here that our implementation is capable of supporting additional measures through the inclusion of appropriate plug-ins. In addition, to support the efficiency of our approach, we have used a similarity pre-calculation mechanism, which ensures that similarity distances between all the concepts of a given ontology are performed only once. Hence, whenever a distance measurement is required this is retrieved from a pre-calculated list.

It thus follows that given a specified set of observations for a process (P) that are semantically associated to the concepts (vc_i) of ontology V , applying $oExp$ generates a set of expanded concepts. If O_c is the set of observation concepts related to the specified observations (i.e. $O_c = \{vc_i : vc_i \text{ is related to } o_i \wedge o_i \in O\}$) then O'_c contains the extended observation concepts:

$$oExp(sD, O_c) = O'_c \quad (3)$$

This set of extended concepts (O'_c) can be regarded as a (i.e. on a semantic level) set of candidate observations that can be taken into account by the process (P). However, this set of candidate observations is improperly defined, as it (O'_c) contains solely the semantic concepts (vc_i) of the candidate observations. This set of candidate observations should be refined so as to identify their syntactic types (t_i) as well, and remove observations (i.e. semantic concepts) that cannot be handled by available actions.

This type of refinement is performed in the following steps of the observation and action expansion process.

A.4.2 Action Expansion

To ensure that the extended set (O'_c) contains observations that can be handled by our system, we need to have actions that can use these observations. The set of actions (A) originally specified for a process (P) points to services that are already defined in the process specification. Nonetheless, these actions may not be able to use the extended set of observations. Therefore, the set of actions should be extended with the inclusion of additional services (i.e. service operations).

To facilitate the inclusion of adequate actions we have introduced a service query mechanism and appropriate heuristics. The prime goal of the query mechanism is to facilitate the discovery of alternative service chains (comprising one or multiple services) that can use as input the extended observations and lead to the achievement of the process goals or sub-goals; sub-goals in this context refers to intermediate states of the specified process (P).

The discovery mechanism uses as input:

- the extended observation concept set (O'_c) and the original set of observations (O),
- the set of known actions (A), i.e. the services already defined in process (P),
- the set of states (S), and

- the process goal states (G).

Based on this input, the query mechanism generates a set of queries that are executed by the Semantic Catalogue Service. The returned results are used for the discovery of service chains Sc , which satisfy our search goals. In the context of our work, each service chain Sc is defined as a finite, ordered set of atomic services (Srv):

$$Sc = \{Srv_i : Srv_0 \prec \dots \prec Srv_{i-1} \prec Srv_{i+1} \prec \dots \prec Srv_n, 0 < i < n \wedge n \in \mathbb{N}\} \quad (4)$$

It should be noted that the produced queries do not contain constraints that enforce strict validation rules i.e. the returned services might not be fully composable, as this is the goal of the integrated planner, i.e. the returned conditional plans contain actions that are fully compatible.

The service discovery process is therefore transformed to a service query formulation problem. Based on the properties of the generic service model defined in [16], a service is a collection of operations offered to its clients via the exchange of appropriate messages over the Web (using appropriate network protocols) at specific endpoints.

Definition A.3. A service is represented as a tuple $Srv = (S_{name}, Ops, Endps)$, where:

- S_{name} is the name of the service,
- Ops is the finite set of supported operations, i.e. $Ops = \{op_i : 0 < i \leq n \wedge n \in \mathbb{N}\}$, and
- $Endps$ the set of available endpoints.

Definition A.4. Each operation $op \in Ops$ is refined as

$$op = \{OP_{name}, Is, Os, C_{pre}, C_{post}, vc\}$$

where:

- OP_{name} is the operation name,
- vc is an ontology concept attributing meaning to a given operation $vc \in V$,
- Is and Os the input and output set of messages, and
- C_{pre} and C_{post} are the supported set of preconditions and postconditions standing prior to and after the execution of the operation, respectively.

Definition A.5. Each message belonging to the input or the output sets i.e. $m \in Is$ or $m \in Os$ is considered as a tuple $m = (M_{name}, t, vc)$ where:

- M_{name} is the name of the message,
- t is the syntactic type and $t \in D$, where D is the set of all supported message types, and
- vc is the related ontology concept, $vc \in V$.

This representation ensures the inclusion of additional functional and/or non-functional properties on the service, operation, and message tuples, if needed. In the context of ENVISION, such additional properties can be the spatial and/or temporal features of the OGC compliant services, e.g. spatial properties of the WFS, SOS, WPS, etc.

To this end, the specification of a service query may contain constraints on any of the specified service properties. Hence, a query q can be formed as a logical conjunction of a finite set of constraints c_j over the specified set of properties. A service query can be formally expressed as:

$$q = c_1 \wedge c_2 \wedge \dots \wedge c_j, \text{ where } 1 \leq j \leq n \text{ and } n \in \mathbb{N} \quad (5)$$

The specified constraints c_j are expressions containing operators such as arithmetic, logical, string, or semantic (e.g. subsumes, consumes) and operands from the set of service properties. Each candidate service's properties should fulfill the specified constraint expressions. The search strategy used for the discovery of additional services will define the constraints that will comprise a service query. Nonetheless, based on the input used by the discovery process and the specified discovery goals, a service query will consist of expressions on the service operation inputs/outputs (i.e. I_s and O_s), their pre- and post-conditions (i.e. C_{pre} and C_{post}), and the semantic concepts (i.e. $vc \in V$).

A.4.3 Service Query Formulation

Among the various search strategies that could be applied [14], backward and forward search seem to be very promising. When using a backward search strategy, the goal is to define service queries that contain specific constraints on the expected output messages (O_s) and the operation's post-conditions (C_{post}), as well as checking whether their inputs contain ontology concepts related to the extended observation set (O'_c). Contrary to that, in a forward search strategy the goal is to define service queries that contain specific constraints on the input messages (I_s), i.e. starting with the ontology concepts of extended observations set (O'_c), and checking if a specific process goal or sub-goal is achieved.

Irrespectively of the applied search strategy, the discovery of services over a registry may return a large number of results. To avoid the generation of a large search space, i.e. considering the set of services contained by the registry and the returned results, we can use constraints posed by the problem definition for the proper formulation of appropriate heuristics.

One of the main requirements of our approach is to provide adaptation steps that reduce the lead to smaller process paths if related information is discovered. We introduce a function $h(s_a, s_b)$ returning the length of the path between states s_a and s_b of a process (i.e. the size of the set of intermediate actions) or a service chain (i.e. the size of the set of comprising service operations):

$$h : (S \times S) \mapsto \mathbb{N} \quad (6)$$

Therefore, given a process P , $h_p(s_0, s_j)$ provides the distance between the starting state s_0 and a goal (or sub-goal) state $s_j \in \{S - s_0\}$ of P . Similarly, if SC is a discovered service chain that accepts as input an extended observation o' and leads to the same state s_j as P , $h_{SC}(s'_0, s_j)$ is the distance between its initial state s'_0 and its goal state s_j . The constraint accruing from the specified requirement can be formally expressed as:

$$h_P(s_0, s_j) > h_{SC}(s'_0, s_j) \quad (7)$$

This constraint can be used as an upper limit of the generated queries and the applied query executions. We need to state here that despite the resulting search process is not complete i.e. we can miss possible solutions satisfying the query constraints but not the length limit, this is not a major drawback, as we are solely interested in cases where a discovered service chain is also leading to smaller processes paths. To further enhance the selection of the most promising candidates and the efficient discovery of solutions, additional heuristics could be used. Nonetheless, the specification of such heuristics depends on the use of additional knowledge from the problem domain.

The outcome of the action expansion step is a finite set of service chains SC , i.e. $SC = \{SC_i : SC_i \text{ discovered service chains, } 0 \leq i \leq n, n \in \mathbb{N}\}$, which use input messages (I_s) with ontology concepts (vc) from the O'_c set. This set of service chains SC constitutes the extended set of actions A' that can be used by the process P .

If $O_{c_{SC}}$ is the set of ontology concepts of all input messages $m \in \cup^{s \in SC} I_{s_s}$ of every service chain $\forall s \in SC$ then:

$$O_{c_{SC}} \cap O'_c \neq \emptyset \quad (8)$$

Nevertheless, to ensure that the extended observations set does not contain observations, which cannot

be handled by available actions, this set should be identical to $O_{c_{SC}}$, i.e. $O_c' = O_{c_{SC}}$. The pruning process used for ensuring the equivalence between these two sets is specified in the following.

A.5 Observation and Action Set Definition

This final step of the observation and action expansion process accommodates the complete specification of the expanded observation variables. More specifically, the identification of the observation variables includes: (i) the pruning of the set of extended observation concepts O_c' , and (ii) the identification of the syntactic type of each extended observation variable and its name.

The pruning process supports the removal of extended observation concepts, which cannot be used by any of the identified service chains. Therefore, the resulting set of extended observation concepts O_c'' contains the intersection of the extended observation concepts and the concepts of the input messages of the discovered service chains:

$$O_c'' = O_c' \cap O_{c_{SC}} \quad (9)$$

Assuming that O_c'' is not an empty set, i.e. $O_c'' \neq \emptyset$, we can associate observation variables to the comprising observation concepts. Based on the given definition, the specification of an observation variable includes its name and syntactic type apart from the associated semantic concept. Given the constraints defined for the expansion of the action set (i.e. the identified service chains should have input messages with semantic concepts from the set of extended observations), we can safely assume that the syntactic type of the related observation variable is the same as the type of the input message of the service chain, which uses this variable.

For the specification of the set of extended observations O' , we define a function returning the set of messages associated to a specific concept:

$$fr : V \times M \mapsto M' \quad (10)$$

Given a set of messages M and a concept vc , $M' = \{m : m \in M \wedge vc_m = vc, \text{ with } vc, vc_m \in V\}$ is the set of messages of M , i.e. $M' \subseteq M$, whose ontology concept vc_m is equal to vc . Similarly, we define a function returning the syntactic type of a given message m :

$$fd : M \mapsto D \quad (11)$$

Therefore, the set of extended observation variables is defined as:

$$O' = \{(O_{name}, vc, t) : \forall vc \in O_c'', t = fd(m) \text{ and } m \in fr(vc, \cup^{s \in SC} I_{s_s}), \forall s \in SC\} \quad (12)$$

I_{s_s} is the set of the input messages of a service chain $s \in SC$. With regards to the extended observation variable names, these can be automatically generated so as to ensure their uniqueness.

The outcomes of this step are twofold, i.e. the extended set of observations and the set of extended actions. The extended set of observations for a given process P is the union of the originally specified (O) and the calculated observations (O'), i.e. $O \cup O'$, while the set of extended actions is the union of the originally specified actions (A) and ones pointing to the discovered service chains (A'), i.e. $A \cup A'$.