



envision
environmental services infrastructure with ontologies

Deliverable D3.3:

MaaS Composition Portal – Version 2

Date:	31 st of December 2011
Author(s):	Roy Grønmo, Sébastien Mosser, Dumitru Roman (SINTEF), Karthikeyan Bollu Ganesh, Alejandro Llaves (UOM)
Dissemination level:	PU (Public)
WP:	3
Version:	2
Keywords:	Service Composition, MaaS Composition Portal, BPMN, Oryx , Model-based mediation, Data mapping, Data transformation, Uncertainty
Description:	MaaS composition portal



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 assuror(s)	Ioan Toma (UIBK), François Tertre (BRGM)
Contributor(s)	All partners

Version history

Version	Date	Description
0.1	31 st of October 2011	Table of Contents
0.2	28 th of November 2011	First full draft
0.3	5 th of December 2011	Full version ready for internal review
0.4	22 nd of December 2011	Updated based on internal review comments.
1.0	31 st of December 2011	Updated based on quality control review

Contents

Executive Summary	3
1 Introduction	5
2 Related Work	6
3 Introducing Mediation in the Composition Portal	8
3.1 Motivations	8
3.2 Integration with Envision Components	9
3.3 ENVISION simple use case: Water Alarm	11
4 Data Schema Information	12
4.1 Schema Information for OGC services	12
4.2 From Schema Information to Ecore Model	12
5 Mediation Portlet	15
5.1 REMICS Mediation Engine Overview	15
5.2 Wrapping the Mediation Engine as a Portlet	16
5.3 Mediation portlet applied to ENVISION's use cases	18
6 From Composition to Execution	20
6.1 Overview	20
6.2 Generating Runtime Data Transformations for OGC-based Compositions	21
6.3 Automated Support for Data Transformations in BPEL	21
7 Uncertainty	24
7.1 Uncertainty-enabled Sensor Observation Services	24
7.2 Visualization of Uncertainty	25
7.3 Integration into the ENVISION Portal	25
8 Summary and Features for Next Release	27
Bibliography	27

Executive Summary

This deliverable reports on the second release of the ENVISION composition portal. The first version of the composition portal was reported by deliverable D3.2 [2]. We do not repeat the things covered there, including design decisions, installation guide, guidelines for modeling BPMN compositions, and how to transform from BPMN to executable artifacts.

The main focus of this deliverable is on the model-based mediation. This enhances the composition portal by providing a user-friendly way of specifying data mediation, which is known to be a complicated and time-consuming task when modeling executable service compositions. We also describe some improvements of the composition portal and our new support for uncertainty management.

1 Introduction

The first release of the ENVISION composition portal is described in ENVISION deliverable D3.2 [2]. This deliverable focuses on the updates since the previous version, with the following main contributions of the second release of the ENVISION composition portal:

1. **model-based mediation** in a semi-automatic tool that helps to specify mappings between source(s) and target data objects;
2. improved versions of **BPMN transformations** to BPEL and WSDL; and
3. **uncertainty management**.

Our composition portal allows the workflow designer to specify integration of existing services into new composite services in a composition tool based on Oryx. A major focus is to make this as user-friendly as possible in order to allow for non-ICT skilled designers to specify compositions. Existing services can be imported into the composition portal from an ENVISION resource component and the low-level properties of this service will be automatically populated in the model.

The existing services are typically provided by different vendors and without plans to be integrated into the new service that a workflow designer intends to build. This means that the output of previously called services will not fit directly with the input of the next service. There can be heterogeneities of different kinds that need to be resolved, such as data representation, semantic and structural differences.

In the previous version of the composition portal, we needed to add data transformations manually in the BPEL file by using XSLT [16] or XPath expressions. In this version we add a model-based mediation portlet that helps the user by suggesting mappings and allowing the workflow designer to refine these mappings. Finally, executable XSLT mappings will be generated and included in the executable artifacts generated from the composition model.

The previous BPMN transformations have been improved by fixing a generation bug in the export format from Oryx, adding a default bindings section to the WSDL file, and by synchronization with work package 6 and the runtime infrastructure. Uncertainty management has been aligned with the UncertWeb project¹ and integrated into the ENVISION infrastructure.

The remainder of this deliverable is structured as follows. Chapter 2 describes related work in the area of model-based mediation since this is the main contribution of this deliverable. Chapter 3 introduces mediation in the composition portal architecture. Chapter 4 covers how to retrieve data schema information for the data objects involved in a mediation. Chapter 5 describes the mediation portlet which is based on a component from the REMICS project. Chapter 6 presents our transformation approach from a composition model to executable artifacts including data transformations. Chapter 7 reports our work on uncertainty management. Finally, chapter 8 gives a summary of our contributions and briefly presents our plans for the next release.

¹<http://www.uncertweb.org/>

2 Related Work

The main new contribution of the composition portal version 2 is about model-based mediation. Here we address related work of model-based mediation in a setting of Web-based compositions.

The challenge of data mapping has been addressed for a number of years and has a long history in the database community. The challenge of resolving heterogeneities is due to differences in syntax, structure, semantics and data representation. With service-oriented architecture and Web service composition, XML has been the standard language to encode data objects. As stated by Nagarajan et al. [9]: “*In the context of Web services, syntactic and model/representational heterogeneities between service message elements are not relevant since the XML based environment automatically resolves them.*” Still, the heterogeneities of structure, semantics and data representation needs to be resolved.

Examples of heterogeneities are:

- data representation. *Example:* Grades are represented as an A-F scale in one property and as a 1-10 scale in another property. Human input is needed to specify a mapping.
- data representation. *Example:* Temperature is represented with the Celsius scale in one property and with the Fahrenheit scale in another property. Can be automated if an algorithm/service is found.
- semantic (synonym). *Example:* ‘Car’ and ‘Automobile’ are two properties with the different name, but that are synonyms for the same concept. Can be detected with a synonym dictionary.
- semantic (homonym). *Example:* Two properties have the name ‘value’, but they are accidentally equal since they represent different concepts. Human input may be needed to avoid that these two properties are incorrectly coupled.
- structural. *Example:* Person/Address/Street in one schema and Person/StreetAddress in another schema. A reasoner can detect such similarities.

BPEL [10] is a widely used language for specifying executable Web service compositions. BPEL comes with two XML transformation languages to define the data mappings. For simple mappings, copy/assign statements with XPath can be used. For more advanced data transformation, BPEL allows to call XSLT files. Although this is expressive enough, Wang et al. [14] argue that writing XPath expressions in copy/assign statements of BPEL is difficult and error-prone even for the IT-professional.

Wang et al. [14] and Nagarajan et al. [9] propose semi-automatic composition. In both approaches the syntactic elements in WSDL/XSD Schema are semantically annotated by SAWSDL [6]. Then, the mapping is specified in relation to the concepts referenced by SAWSDL. This requires so-called *lifting* and *lowering* algorithms. A lifting algorithm transforms the source XML document into SAWSDL referenced concepts, while a lowering algorithm transforms back from SAWSDL referenced concepts to target XML documents.

Waworuntu and Bailey [15] have a tool, called XSLTGen, that takes a source and target XML document as input and produces an XSLT to perform a transformation from source to target. This requires that we produce the mapped target XML document from a source XML document. When composing new services, such corresponding documents is non-existing. Hence, in order to use XSLTGen we need to perform this manually and that can be a complicated task.

Wu et al. [17] have a matching process that looks for synonyms and structural similarities. Their approach is fully automatic, which means the best match, as guessed by the engine, is always used. We believe that this fully automatic approach will fail in many typical scenarios, and hence it cannot be used as a general solution.

Mocan and Cimpian [7] propose a semi-automatic matching process to specify mappings between a source and target ontology. A matching process will first use synonyms and structure information to suggest mappings for a user which then refines the mapping at design-time. The specified mapping at design-time is used to create fully automatic mappings between ontology instances at run-time. The approach only maps between so-called WSMO ontology instances. Hence, in order to be used for Web service data mediation there is a need to map from source XML documents to WSMO ontology concepts, and finally there is a need to map the mapped WSMO ontology concepts to XML documents. Such mappings between WSMO ontology concepts and XML documents are not part of the given framework.

3 Introducing Mediation in the Composition Portal

The intention with the composition portal is to provide a Web-based modeling environment where a workflow designer can reuse existing services in order to create a new composite service. The main extension of the composition portal compared with the previous version is the inclusion of a mediation engine.

The model-based mediation component is a key enabler to support interoperability between heterogeneous services.

3.1 Motivations

Even if there is no real consensus on the definition of service-oriented architecture (SOA) in the general case, the following definition was given by Dodani in 2004 and is now commonly accepted: “SOA enables flexible integration of applications and resources by: (1) representing every application or resource as a service with standardized interface, (2) enabling the service to exchange structured information (messages, documents, “business objects”), and (3) coordinating and mediating between the services to ensure they can be invoked, used and changed effectively” [1]. One can clearly see the intrinsic interest for mediation in this context (key point #3). In this context, we defined an engine that support such a mediation. Assuming that the description of the message exchanged by two services are described using a machine-readable data structure, our objective is to define a generic engine able to identify similarities in the existing structures.

Mediation is defined by the Wordnet¹ reference thesaurus as: “a negotiation to resolve differences that is conducted by some impartial party”. In ENVISIONs SOA setting, mediation is needed to produce the correct input data object for a service based on the output data object(s) produced by calls to previous service(s). A mediation process will result in resolved differences, that are materialized as *mappings*. A mapping defines a relation between source and target elements. Such a mapping can be at a high-level where it means that two concepts are somehow related, or it can be very precise. A precise mapping is called an executable specification, if it is written in a language that can be automatically processed by a tool. A complete set of precise mappings are typically called a *transformation* specification. There are a variety of languages that can be used to specify transformations, including the general purpose language Java, and the XML transformation language XSLT.

The mediation process can be tool-supported (called mediation tool/engine), normally as a semi-automatic tool where user input is required to define a complete and precise mapping. Such a mediation tool can try to automatically propose mappings based on metadata descriptions of objects to be mediated.

¹<http://wordnet.princeton.edu/>

3.2 Integration with Envision Components

The composition portal consists of three internal parts: Oryx (composition portlet) for BPMN-based composition modeling, mediation portlet that executes the mediation engine and an underlying database (PostgreSQL) for persistent storage. Figure 3.1 shows the composition portal and its main interactions with two other ENVISION components:

- *Resource Portlet [5]*. This is developed in work package 2. It contains all the available resources as OGC-based and traditional Web services. The user selects a service at a time and import it into the composition portal. It results in the creation of a new task in the BPMN model. This task gets all the required properties assigned from the resource portlet so that one can generate executable artifacts from the composition model.
- *Execution Infrastructure [13]*. This is developed in work package 6. It supports the execution of the modeled compositions. When a BPMN composition has been finalized it is exported as an executable artifact which is sent to the execution infrastructure. It is also stored in the resource repository, as a newly available resource. At the technical level, the environment produces BPEL and WSDL files which are sent to the BPEL engine for deployment and execution.

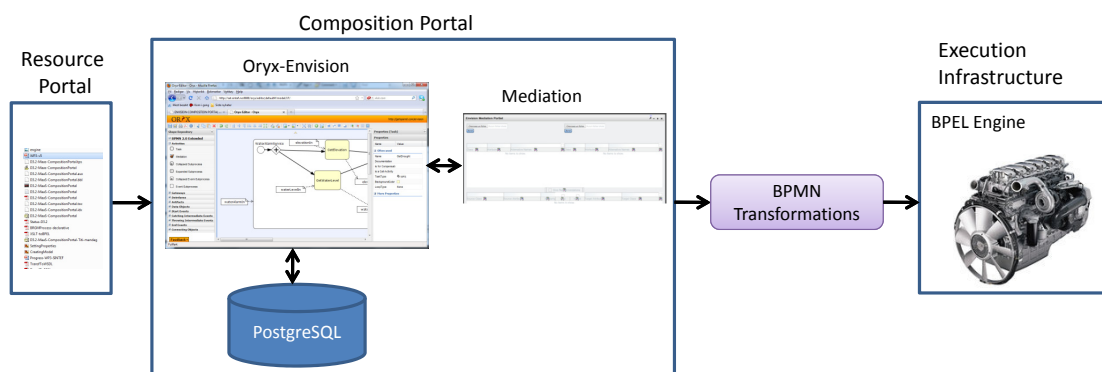


Figure 3.1: Interaction between the composition portlet and other ENVISION components

Figure 3.2 describes the sequence of events that occur when a modeler initiates the specification of a model-based mediation. One specific mediation will involve exactly one target data object and one or more source data objects that are used to produce the target. The process proceeds sequentially through the six steps described below.

- **Step 1.** The modeler identifies the data objects involved in the mediation. One or more source data objects are mapped to exactly one target data object. It is assumed that there are data flows from every source object to the target object in the BPMN model. This means that it is sufficient to select the target data in order to deduce the involved source data objects.
- **Step 2.** The composition portlet retrieves the metadata of the data objects, which is extracted from the WSDL and RDF files of the corresponding services in the resource portlet.

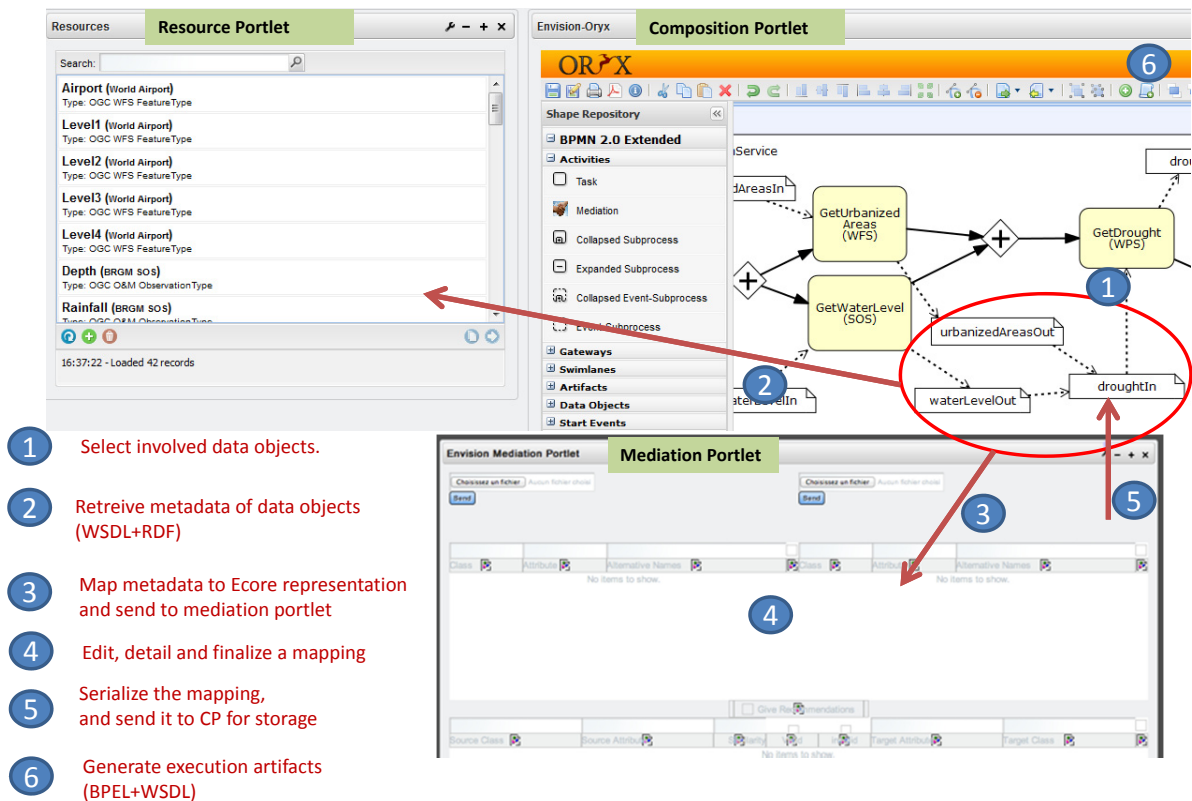


Figure 3.2: Model-based mediation activated from the composition portlet

- **Step 3.** The mediation portlet requires that the metadata about the involved data objects are represented in the Ecore format, which is the Eclipse metamodel standard. Hence, we transform the metadata representations into Ecore in this step. Further details are given in chapter 4.
- **Step 4.** The mediation portlet performs a matching process of the received Ecore source(s) and target. The outcome is a ranked list of possible mappings from source(s) to the target. The user then selects which of the ranked mappings are correct and possibly details the mappings. There may also be a need to add mappings not suggested by the mediation portlet.
- **Step 5.** When the user has finalized the mapping, a store menu item is pressed. This results in a serialization of the mapping which is sent back to the composition portlet (CP) and stored as a property of the target data object. Further details are given in chapter 5.
- **Step 6.** When the composition model is finalized, including specified mappings in all relevant places, execution artifacts can be generated. One BPEL file and one WSDL file is generated for the new composite service. The generation includes one XSLT file for each specified mapping, and these XSLT files are called from the BPEL file. Further details are given in chapter 6.

Mediations are defined by the data flows. One output data can have multiple outgoing data flows to other data objects, which implies that one source shall be mapped to several targets.

An input data object can have multiple incoming data flows from multiple data objects (as in the example described in the subsection below), which implies that multiple sources are mapped to one target. Hence, our framework supports n-to-m mappings.

3.3 ENVISION simple use case: Water Alarm

We will use the ENVISION simple use case as a running example. This service has been modeled as a BPMN model that calls three services in order to achieve its task of indicating a water alarm concerning the supply of drinking water.

We consider here the composition expressed in the context of the simple use case. To model such a composition, we use the BPMN notation, as described in 3.3. At the technical level, we rely on the three following services²:

- A WFS service delivering urbanized areas,
- an SOS service delivering the water level of subsurface water bodies, and
- a WPS service consuming water levels and urbanized areas, intersecting the geometries, and producing a new vector layer presenting a classification of drinking water supply of the urbanized areas.

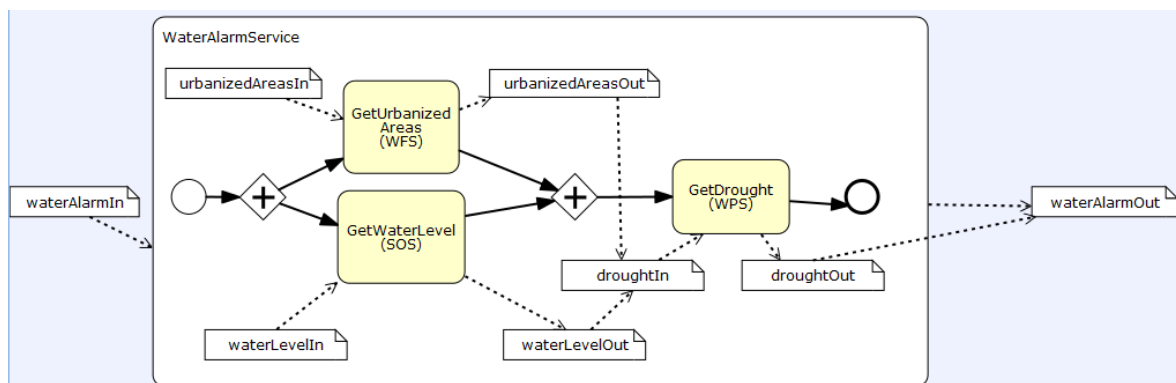


Figure 3.3: Simple Use Case composition, expressed in BPMN

The composition model retrieves data from the SOS and WFS services in parallel, as there is no correlation between these two entities. The WFS service returns data about water level (`waterLevelOut`) and the SOS service returns data about urbanized areas (`urbanizedAreasOut`). The information from these two data sources are combined into a new data object, called `droughtOut`, which is the input data object of the WPS service.

In a composition model, there is potentially a need for data mediation for each input data object. In our example, data mediation is needed in order to know how the source data objects (`waterLevelOut` and `urbanizedAreasOut`) shall be mapped to the target data object (`droughtOut`). The goal of the mediation portlet is to help a non-ICT skilled workflow designer to specify the needed mappings.

²<http://kenai.com/projects/envision/pages/SimpleUseCase>

4 Data Schema Information

The previous chapter described how the composition portal works from the user point of view. However, things happen behind the scene to support the previously described scenario.

This chapter describes how to retrieve information about the data schemas for the involved data objects. A data schema for a data object defines the expected elements and how they are structured.

4.1 Schema Information for OGC services

Web services rely on three technologies to support the implementation of SOA: (i) XML SCHEMA (XSD) to describe the data schema for an XML document, (ii) WEB SERVICE DESCRIPTION LANGUAGE (WSDL) to describe service interfaces, and (iii) SIMPLE OBJECT ACCESS PROTOCOL (SOAP) for the message exchange protocol.

With Web services, the data objects are XML documents with *elements* and *attributes* as the two building blocks. A WSDL file defines the service interface and uses XML Schema to define the data schemas for the input and output of a Web service operation. Here, the allowed element and attribute names and their nesting structure is defined.

Additional meta-data about the data objects may be defined as semantic annotations with links to ontology concepts, logical expressions etc. There are numerous ways of doing this and numerous languages in which to capture such information. In ENVISION we focus on using OGC-based services. Such services are also WSDL-based services, but where the interface and the set of operations are fixed. The data structure of the service, including each of its inputs and outputs, is described in WSDL and often also using ontology representation (RDF/POSM standards).

This chapter describes how data schemas are extracted from the available artifacts used in the composition portal. These data schemas are then sent to the mediation portlet to support the definition of a mediation between a given set of heterogeneous services. We need to convert the data schemas into so-called Ecore models¹ since this is the required input format of the mediation portlet from the REMICS project.

An Ecore model is defined by classes, attributes, references and data types. A class can have attributes and references. An attribute has a name and a data type, and a reference defines an association between two classes.

4.2 From Schema Information to Ecore Model

For WSDL-based services we use an Eclipse plugin (named `xsd2ecore`²) to retrieve an Ecore model that includes the information about the data object we are interested in. In the first version we use the entire Ecore model of the service. This model will then include parts

¹<http://www.eclipse.org/modeling/emf/?project=emf>

²<http://www.eclipse.org/modeling/emf/docs/overviews/XMLSchemaToEcoreMapping.pdf>

that are not part of the mediation scenario, such as inputs/outputs of other operations and also the input/output parameter of the relevant operation since we only use either the input or the output data object in the mediation scenario. In the next version we consider to prune this Ecore model so that only the relevant information about the mediation data object is kept.

Our default mapping to Ecore uses the WSDL file, and this is sufficient for the SOS and WFS services. However, for the WPS input data object this does not provide sufficient information. A WPS input data object provides special sections with identifiers to define the expected inputs in the associated RDF/POSM file. This means that we have to parse the RDF/POSM file to retrieve this essential information. In fact, for the WPS input data, these identifiers represent all the information we need in order to specify the mapping. We have developed an XSLT script to produce an Ecore model from the RDF/POSM file describing a WPS. Hence, for WPS input data objects we retrieve the RDF/POSM file of the WPS service and call our own XSLT script to produce Ecore. For all other services, we will map the WSDL file to Ecore.

In the simple use case example, a mediation is needed between the output provided by the SOS and WFS services and the input required by the WPS service. Figure 4.1 represents the obtained Ecore models which are the targets of our transformations from WSDL/XSD and RDF/POSM files. The figure shows the complete Ecore model the WPS input. For the other two Ecore models we show only an extract. The entire models are large since they are produced from XML Schemas that import several other XML Schemas.

The correct mapping for this use case consists of two relations:

- the source element `ObservationCollectionType` maps to the target element `ObservationCollectionInput`, and
- the source element `FeatureCollectionType` maps to the target element `FeatureCollectionInput`.

The next chapter shows how the mediation portlet helps to identify the correct mapping.

5 Mediation Portlet

The mediation portlet is essentially a mediation engine component that has been wrapped as a Web-based portlet to be integrated into the portlet-based environment of the ENVISION infrastructure. The mediation engine is initially designed by SINTEF in the framework of the REMICS project (EU FP7, contract number 257793). The REMICS project focuses on the migration of existing application to cloud-computing providers. Mediation techniques are needed in REMICS to properly ensure the interoperability between legacy systems and new infrastructures.

This chapter describes how the REMICS mediation engine has been adopted to help us achieve a mediation for integration of OGC and WSDL-based services from different vendors. The ENVISION use cases serve the purpose of realistic test cases for the REMICS engine.

5.1 REMICS Mediation Engine Overview

This section gives a short overview of the mechanisms provided by the mediation engine. The interested reader may refer to the REMICS deliverable and web page¹ that describe this engine² for an in-depth and detailed description.

In the general case, fully automated mediation implies a high trust in the engine that supports its implementation. To tackle this intrinsic challenge, a semi-automatic mediation engine is usually preferred instead of a fully automated mediation. Based on heuristics, these systems analyse a set of given inputs, and compute some mapping suggestions to support the actual mediation. For example, where a fully automated mediation engine should identify that an element $e \in E_1$ match an element $e' \in E_2$, a semi-automatic mediation engine will compute that “ e and e' are similar”. Then, it is up to the user to follow the mapping suggestion given by the engine. Similarity can be expressed as a real value that belongs to a given interval (e.g., $\in [0, 1]$, where higher values indicate stronger degree of similarity), where a match is a canonical value (i.e., $\in \{true, false\}$). Looking at the problem from a logical point of view, one can recognize the differences between predicate logic and fuzzy logic.

The global process is described in figure 5.1, by UML 2.0 activity diagrams, with a so-called “human-in-the-loop” approach. The workflow designer selects the source(s) and target data objects in the composition portal and activates the mediation. This triggers a retrieval of source(s) and target schemas which are sent to the mediation engine.

The mediation engine preprocess these schemas which result in schemas with additional meta-data. These meta-data annotations can be based on expressive elements such as semantic descriptions (e.g. ontologies, with triples such as “Book is a Publication”), or basic elements such as synonyms dictionary (e.g., “Book is synonym of Volume”) or abbreviation index (e.g., “Vol. means Volume”).

¹<http://remics.eu>

²In REMICS, the mediation engine is called *recommendation engine*

Currently, we produce schemas with synonym relations. This means that we get a fairly powerful matching and reasoning also for basic WSDL-services which are not semantically annotated. Such non-annotated services are very common in practice. Future versions of the mediation portlet may take advantage of semantic annotations, such as WSMO, RDF, OWL-S, so that the matching process becomes even more precise.

The workflow designer is then asked to validate or refine the given mapping suggestions. When the workflow designer has finalized the mappings, a mapping table is exported and the mediation process ends.

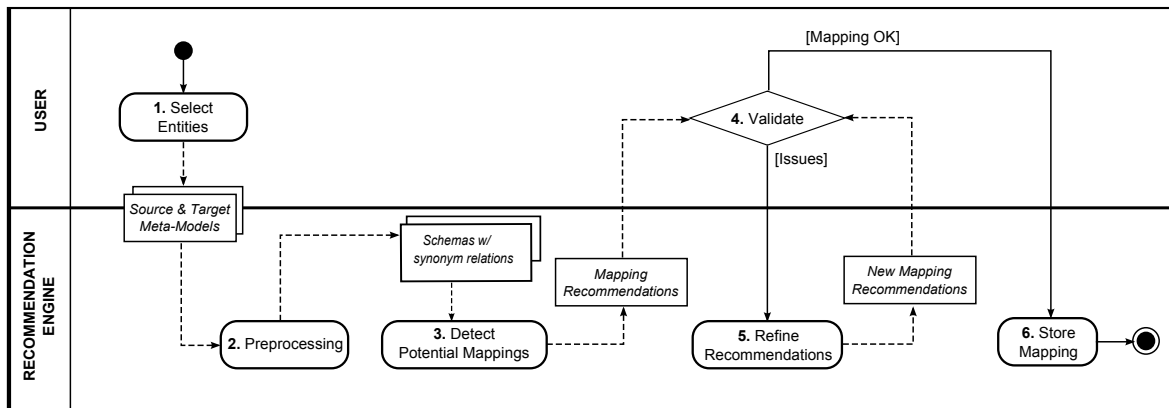


Figure 5.1: Mediation process

At the technical level, the engine is implemented using the Scala language [11] (object-oriented and functional paradigms). It intensively relies on the Actor paradigm [4] to support the parallelisation of the computation.

5.2 Wrapping the Mediation Engine as a Portlet

The ENVISION portal relies on the *portlet* technology at the integration layer. Thus, the engine is exposed as a *portlet* to support its complete integration in the ENVISION tool suite. To support such an integration, we used the Google Web Toolkit³ to implement a web-based interface, exposed as a *portlet*. A screenshot of this interface is depicted in figure 5.2.

On the top, the interface is divided in two. Each side list the elements found in the given meta-models, with the associated meta-data (“Alternative Names” column). When the user clicks on the “Give Recommendation” button, the engine computes the similarities based on the given source(s) and target models, and express a similarity score $s \in [0, 1]$. A similarity of 1 means that the two elements seem to be fully equivalent from a structural and semantic viewpoint, and a similarity of 0 means that the mediation engine does not detect any similarities between the two elements. The user is asked to identify valid or invalid mapping suggestions (ticking the right check box). The “Refine Recommendation” button is used to trigger a matching process, that recompute the similarities based on the new information given by the user.

³<http://code.google.com/intl/en-EN/webtoolkit/>



Figure 5.2: GWT interface used to interact with the engine

5.3 Mediation portlet applied to ENVISION's use cases

This section shows how the mediation portlet is used to support the mediation between services in the context of ENVISION use case.

5.3.1 Using the mediation portlet

For the sake of conciseness, we only expose here the suggested mappings of the mediation portlet as plain text. These data are more concise for a paper-based description than the ones obtained through the mediation portlet. Using the SOS and WFS output data structure on the one side, and the WPS on the other side, the engine automatically produces the following suggested mappings:

```
mapping SOS_WFS_Vs_WPS {
  WPS::ObservationCollectionInput --> SOS::ObservationCollectionType [0,6024]
  WPS::FeatureCollectionInput --> WFS::FeatureCollectionType [0,6901]
  ecore::EFeatureMapEntry --> ecore::EFeatureMapEntry [1,0000]
  ecore::EStringToStringMapEntry --> ecore::EStringToStringMapEntry [1,0000]
  ecore::EObject --> ecore::EObject [1,0000]
  type::AnySimpleType --> type::AnySimpleType [1,0000]
  type::AnyURI --> type::AnyURI [1,0000]
  type::Boolean --> type::Boolean [1,0000]
  type::Date --> type::Date [1,0000]
  type::DateTime --> type::DateTime [1,0000]
  type::Decimal --> type::Decimal [1,0000]
  type::ID --> type::ID [1,0000]
  type::Integer --> type::Integer [1,0000]
  type::NonNegativeInteger --> type::NonNegativeInteger [1,0000]
  type::PositiveInteger --> type::PositiveInteger [1,0000]
  type::QName --> type::QName [1,0000]
  type::String --> type::String [1,0000]
  UnknownType --> UnknownType [1,0000]
  UnknownClass --> UnknownClass [1,0000]
}
```

The two first lines represents the core of the suggested mappings. On the first line, the engine suggests to fill the `ObservationCollectionInput` part of the `WPS` service with the `ObservationCollectionType` output of the `SOS` service. On the second line, it suggests to fill the `FeatureCollectionInput` of the `WPS` service with the `FeatureCollectionType` of the `WFS` service. In front of these suggestions, the workflow designer express that he/she validates the suggested mapping. The other lines indicate full equivalence (similarity equals to 1) between the mapped elements. These elements are common classes and data types present in all Ecore models, which are not essential for the mapping specification.

5.3.2 Exporting Mapping to Support Automation of Executable Transformations

An XML serialization of the mapping specification is shown below:

```
<?xml version="1.0" encoding="utf-8"?>
<mediation>
  <inputs>
    <file>SOSurbanizedAreasOut.xsd</file>
```

```

    <file>WFSwaterLevelOut.xsd</file>
  </inputs>
  <outputs>
    <file>WPSdroughtIn.xsd</file>
  </outputs>
  <elements>
    <element score="0.6024">
      <source file="SOSurbanizedAreasOut.xsd">SOS::ObservationCollectionType</source>
      <target file=">WPSdroughtIn.xsd">WPS::ObservationCollectionInput</target>
    </element>
    <element score="0.6901">
      <source file="WFSwaterLevelOut.xsd">WFS::FeatureCollectionType</source>
      <target file=">WPSdroughtIn.xsd">WPS::FeatureCollectionInput</target>
    </element>
  </elements>
</mediation>

```

In ENVISION, we generate executable data transformations (as XSLT code) based on the XML serialization of the specified mapping. These executable XSLT transformations are used at runtime to accurately support the expected data exchange. The next chapter provides further details on how these data transformations are called from within the execution language of BPEL.

6 From Composition to Execution

The previous chapters describe how a mediation engine is used to tame the complexity of web services composition. We describe here how a composition model is transformed into a set of executable artifacts to be used at runtime.

6.1 Overview

Figure 6.1 shows the artifacts involved in the process from modeling the composition to the generation of WSDL and BPEL as the Web service artifacts. We have implemented several plugins for the Oryx BPMN editor to support the described approach. The Oryx implementation has an exporter that generates DI XML [12], which is the standard XML format for BPMN models. For each extension to the modeling environment of Oryx, we extended the DI XML exporter with the additional elements and properties. From a technical point of view, we have to pre-process the exported model to fix some inconsistencies in the retrieved representation (see “limitations” at the end of this chapter). Based on the fixed DI XML representation of the composition model, we have implemented two XSLT-code transformations to generate the following runtime artifacts:

- An executable process description of the composition as a BPEL file, and
- An interface description of the composite service as a WSDL file.

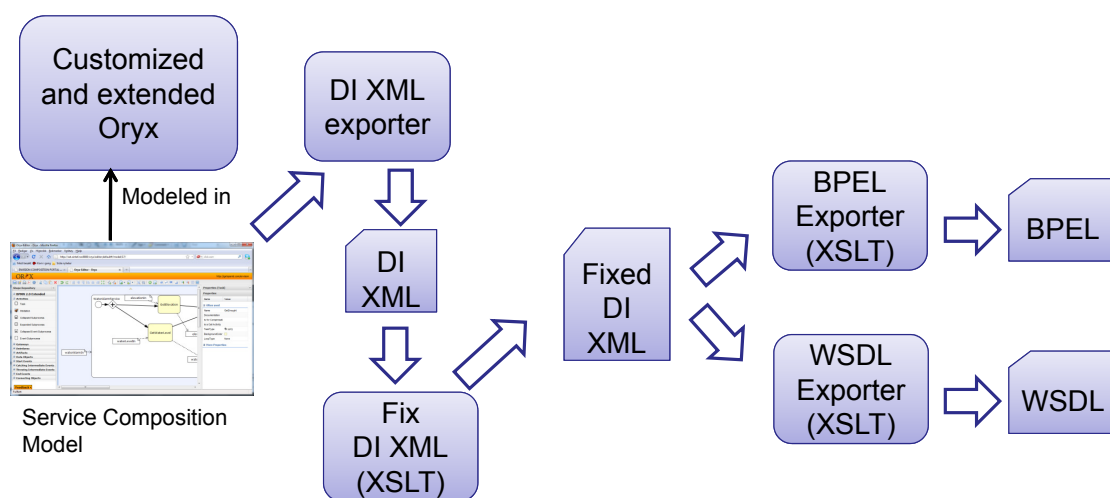


Figure 6.1: Generating execution artifacts (BPEL and WSDL) from the composition portlet

6.2 Generating Runtime Data Transformations for OGC-based Compositions

With OGC-based service compositions, the target service of a mediation is normally a WPS service. This is because WPS is the only OGC service designed to process arbitrary data objects produced by other services. This section describes how we can generate data transformations for OGC-based compositions where the target service is a WPS service.

To support the execution of the specified mappings, we rely on the XSLT transformation technology. We have implemented an XSLT script which is a higher-order transformation script in that it transforms the input into another XSLT script. The higher-order XSLT (Figure 6.2) transforms the serialized mapping description into XSLT code to handle the runtime data transformation. At design-time, the mediation portlet is used to produce a mapping specification captured in an XML file (`Mapping.xml`). This mapping file and the higher-order XSLT (`GenXSLT.xsl`) is processed by an XSLT processor to produce the XSLT to be used at runtime to handle the data mediation.

In the simple use case, the runtime XSLT (`MakeWPSInput.xsl`) takes the two outputs from the SOS service and the WFS service, respectively, as its inputs and produces the WPS input XML (`getDroughtIn.xml`). The WPS input document consists of boilerplate XML and fixed positions with identifiers `observationCollection` and `featureCollection`, to define its key input parts.

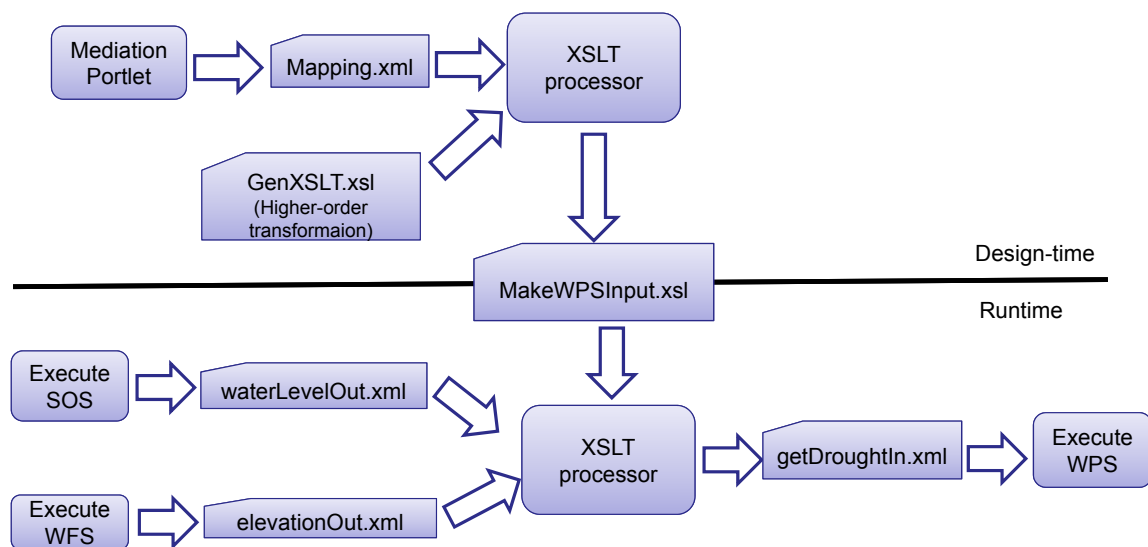


Figure 6.2: Generating XSLT for data mediation based on higher-order XSLT

6.3 Automated Support for Data Transformations in BPEL

We have written XSLT code (described in ENVISION Deliverable D3.2) that takes DI XML as input and produces BPEL and WSDL. The previous generation of WSDL has been extended to also include a so-called *bindings* section in this version of the composition portal. The bindings section describes the exact protocols to use when invoking the Web service.

envision

The generation assumes that the implementation of the service uses SOAP over HTTP using so-called *literal* encoding style.

When the composite model has been finalized with mapping specifications defined for all input data objects, we can generate BPEL. The BPEL generation process is illustrated by a UML 2 sequence diagram in Figure 6.3. The focus here is on how to integrate the XSLT code into BPEL.

There is a loop construct to ensure that we generate XSLTs for each input data object. We retrieve the mapping specification of the input data object and generate the XSLT from it. All of this happens in the composition portlet. Then the XSLT is stored in the resource portlet and the composition portlet receive the URL of the XSLT from the resource portlet. The URL of the XSLT is stored with the data object.

When we have stored XSLT URLs for each input data object, we are ready to generate the complete BPEL. The generation of BPEL is as before (described in ENVISION deliverable D3.2) with the addition of calling XSLTs in the correct positions for each data mediation needed.

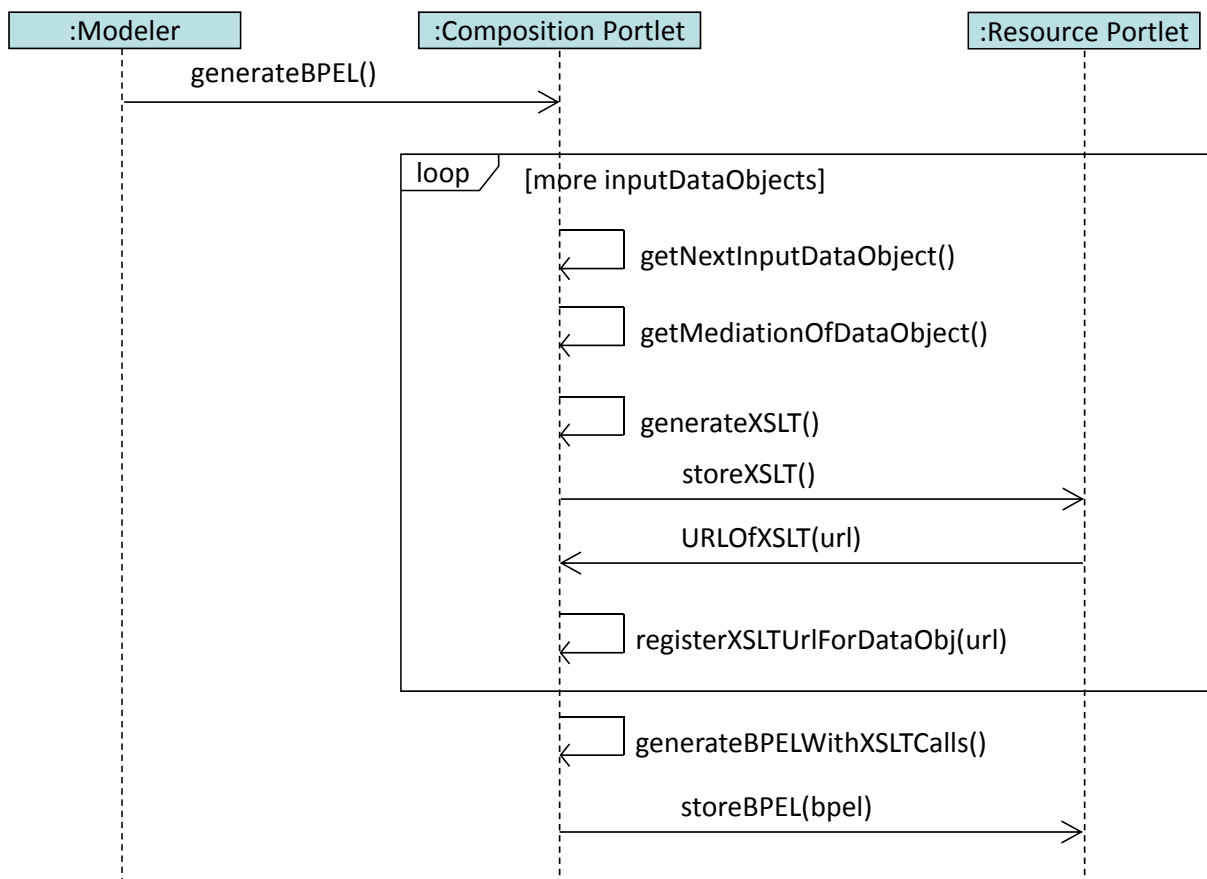


Figure 6.3: Interactions in the process of generating BPEL with XSLT-based transformations

Limitations & Workaround. Unfortunately, the DI XML exporter has a few bugs which we have reported back to the Oryx open source community. Data flow associations between data objects are not exported and all objects inside a subProcess are duplicated. In our first versions of the transformation, we needed to remove the duplicate elements manually before

finalizing the transformation. In this version we have implemented an intermediate XSLT that fixes the problem of duplicate entries. This XSLT is called behind the scenes and the workflow designer does not need to be aware of this fix. As a workaround to the missing data flows, we should currently select all the involved data objects in the data mediation. The output data object(s) selected become source(s), while the input data object becomes the target of the mediation.

7 Uncertainty

In ENVISION deliverable D3.1 [3] we introduced basic concepts on uncertainty modelling together with current encoding standards, the role of uncertainty in service compositions, and our approach to integrate uncertainty management in ENVISION. This section describes the outcomes of our collaboration with the UncertWeb project and the integration of the results into the ENVISION infrastructure.

It is important to highlight that the uncertainty visualization tool presented below will not be included in the MaaS Composition Portal. We assume that the sensor data service responses contain uncertainty annotations derived from the physical sensor's error and encoded as UncertML. Sensor observations are requested at runtime whereas the service composition occurs at design time and involves service descriptions instead. Therefore, the use of the uncertainty visualization will only be addressed to evaluate how uncertain the data provided by a service is before including that service into the composition.

The uncertainty related to the result of an environmental service composition can be visualized with our application if i) the output consists of observations encoded as O&M with uncertainty annotations in UncertML (version details in section 7.1), and ii) the uncertainty parameters are propagated in a proper manner along the data mediation steps of the composition. At this stage of the project, the propagation of uncertainty through the service composition has not been addressed. This issue depends on the results of the collaboration with the UncertWeb project, as described in D3.1.

7.1 Uncertainty-enabled Sensor Observation Services

The migration of environmental models to Web service compositions is one of the main purposes of our project. Sensor data services are usually the input of these service compositions and, therefore, the entrance point of uncertain data into the model. To assess the uncertainty present in Sensor Observation Services (SOS) [8] and prevent propagation we provide a Portlet which allows visualizing the measurement error implicit in the observation.

UncertML is a conceptual model and XML encoding designed for encapsulating probabilistic uncertainties¹. UncertML 2.0 will be used to enrich SOS data responses - encoded in Observations and Measurements (O&M) 2.0 - with uncertainty metadata. Figure 7.1 shows an example of a Gaussian distribution description in UncertML which is embedded in a SOS response.

A JavaScript application was developed to parse the getObservations response of SOS. By using the GET method of OpenLayers, the document containing the observations and the uncertainty metadata is requested from the service.

¹<http://www.uncertml.org/>

7.2 Visualization of Uncertainty

Nowadays, uncertainty visualization is a hot research topic. Depending on the data source and the type of uncertainty, various methods can be used to depict the information, e.g. colour models, charts or graphs. In the case of our prototype, where a continuous flow of underground water level observations are depicted, we selected a time-series chart. Figure 7.2 shows a snapshot of the UncertaintyViewer. The mean values are expressed as red points and connected with a line. The Max and Min points in blue represent the maximum and minimum values that each observation can reach, i.e. the variance.

7.3 Integration into the ENVISION Portal

The UncertaintyViewer is integrated with the ENVISION MapPortlet (see figure 7.3) into the Uncertainty Portlet. When a user selects a sensor point in the map, the values of the

```
<om:resultQuality>
  <gmd:DQ_QuantitativeAttributeAccuracy>
    <gmd:result>
      <gmd:DQ_UncertaintyResult>
        <gmd:value>
          <un:GaussianDistribution>
            <un:mean>0.89 0.87 </un:mean>
            <un:variance>0.01 0.01</un:variance>
          </un:GaussianDistribution>
        </gmd:value>
      </gmd:DQ_UncertaintyResult>
    </gmd:result>
  </gmd:DQ_QuantitativeAttributeAccuracy>
</om:resultQuality>
```

Figure 7.1: Gaussian distribution metadata encoded in UncertML in a SOS response.

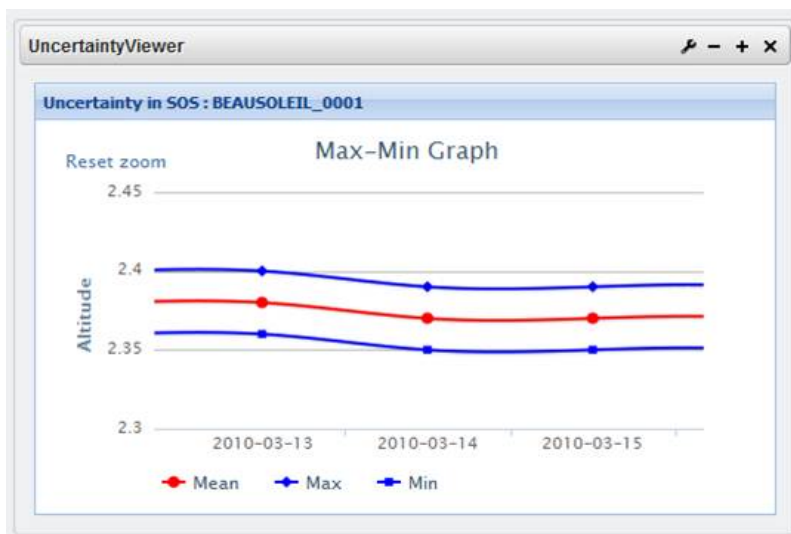


Figure 7.2: Gaussian distribution metadata encoded in UncertML in a SOS response.

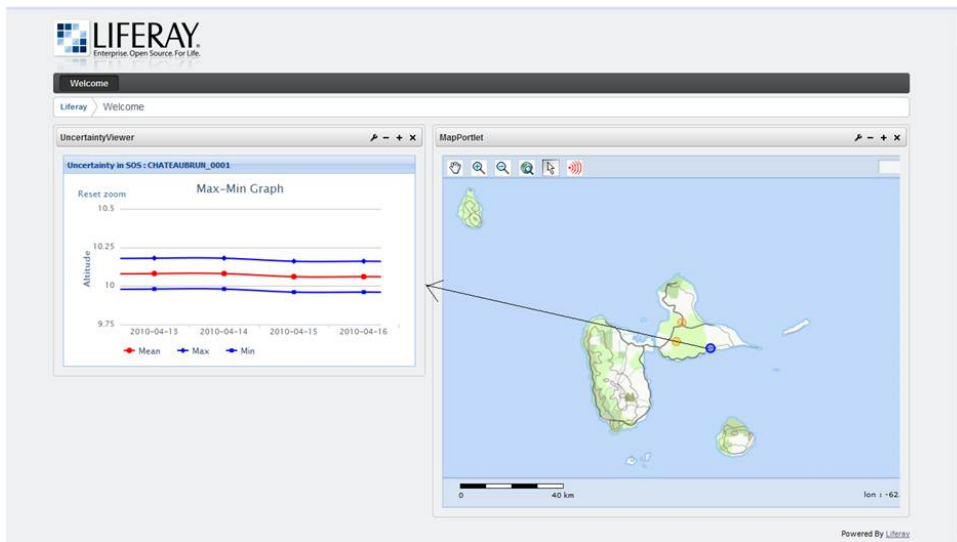


Figure 7.3: UncertaintyViewer (left) displaying observations from the sensor selected at the MapPortlet (right).

last observations are displayed in the chart surrounded by the confidence intervals (in blue). Future work will focus on extending the UncertML parser to support additional distributions and on including more visualization possibilities.

8 Summary and Features for Next Release

This deliverable provided an overview of the second release of the composition portal. The main new achievement in this second release is a model-based mediation portlet (most parts are implemented, but they are not integrated yet). We have also updated the BPMN-based transformations to BPEL and WSDL, and we have a running server with the resource portlet and the composition portal working together at <http://set.sintef.no:8082>. Some support for uncertainty management has been integrated into the ENVISION infrastructure.

Even though much work has been done in the area of data mediation, we have not seen other approaches to model-based mediation with focus on OGC services. We provide specialized support to define mediation for OGC-based service compositions. By using a mediation engine from the REMICS project to suggest mappings based on structures and synonym dictionaries, we are able to reduce the effort needed to define mapping specifications. Our behind-the-scenes generation of XSLT means that low level technical details are transparent for the workflow designer and is a step forward to allow non-ICT skilled users to model service compositions. The model-based mediation support will be fully integrated into the Web-based infrastructure of ENVISION.

The next release of the composition portal will be provided at M34. In that release, we will integrate all the parts of the model-based mediation portlet and extend the BPEL generation with calls to the mapped XSLT transformations for data mediation, as described in this deliverable. Furthermore we intend to model and try out the composition portal on the landslide and oil spill use cases. For these use cases we will investigate how we can introduce extension points for more advanced data mediation scenarios than one-to-one correspondences.

Bibliography

- [1] Mahesh H. Dodani. From Objects to Services: A Journey in Search of Component Reuse Nirvana. *Journal of Object Technology*, 3(8):49–54, 2004.
- [2] Roy Grønmo, Dumitru Roman, Luis Costa, and Ioan Toma. Deliverable D3.2: MaaS Composition Portal Version 1 Initial Development and User Guide. <http://www.envision-project.eu/deliverables/>.
- [3] Roy Grønmo, Dumitru Roman, Alejandro Llaves, and Patrick Maué. Deliverable D3.1: MaaS Composition Portal Architecture Specification. <http://www.envision-project.eu/wp-content/uploads/2010/01/D3.1-FINAL.pdf>.
- [4] Philipp Haller and Martin Odersky. Scala Actors: Unifying thread-based and event-based programming. *Theoretical Computer Science*, 410(2-3):202–220, 2009.
- [5] Florian Husson, Francois Tertre, Joel Langlois, Roy Grnmo, Iker Larizgoitia, Patrick Maue, and Miha Grcar. Deliverable D2.2: Environmental Semantic Web Portal Version 1 Initial development and user guide. <http://www.envision-project.eu/deliverables/>.
- [6] Jacek Kopecky, Tomas Vitvar, Carine Bournez, and Joel Farrell. SawSDL: Semantic annotations for WSDL and XML schema. *IEEE Internet Computing*, 11:60–67, 2007.
- [7] Adrian Mocan and Emilia Cimpian. An ontology-based data mediation framework for semantic environments. *Int. J. Semantic Web Inf. Syst.*, 3(2):69–98, 2007.
- [8] A. Na and M. Priest. *OGC Implementation Specification 06-009r6: OpenGIS Sensor Observation Service (SOS)*. Open Geospatial Consortium, 2007.
- [9] Meenakshi Nagarajan, Kunal Verma, Amit P. Sheth, and John A. Miller. Ontology driven data mediation in web services. *Int. J. Web Service Res.*, 4(4):104–126, 2007.
- [10] OASIS. Web Services Business Process Execution Language Version 2.0. Technical report, OASIS, 2007.
- [11] Martin Odersky, Philippe Altherr, Vincent Cremet, Iulian Dragos, Gilles Dubochet, Burak Emir, Sean McDirmid, Stéphane Micheloud, Nikolay Mihaylov, Michel Schinz, Lex Spoon, Erik Stenman, and Matthias Zenger. An Overview of the Scala Programming Language (2. edition) lamp-report-2006-001. Technical report, École Polytechnique Fédérale de Lausanne (EPFL), Switzerland, 2006.
- [12] OMG. Business Process Modeling Notation (BPMN) Version Beta 1 for version 2.0 - OMG Document Number: dtc/2009-08-14. <http://www.omg.org/spec/BPMN/2.0/>, August 2009.
- [13] A. Tsalgatidou, G. Athanasopoulos, I. Pogkas, and P. Kouki. Deliverable D6.2: ENVISION Adaptive Execution Infrastructure Version 1 Initial development and user guide. <http://www.envision-project.eu/deliverables/>.

- [14] Zhiming Wang, John A. Miller, Jessica C. Kissinger, Rui Wang, Douglas Brewer, and Cristina Aurrecochea. WS-BioZard: A Wizard for Composing Bioinformatics Web Services. In *IEEE Congress on Services, Part I, SERVICES I*, pages 437–444. IEEE Computer Society, 2008.
- [15] Stella Waworuntu and James Bailey. XSLTGen: A System for Automatically Generating XML Transformations Via Semantic Mappings. *J. Data Semantics V*, pages 91–129, 2006.
- [16] World Wide Web Consortium (W3C). XSL Transformations (XSLT) Version 1.0. <http://www.w3.org/TR/xslt>, 1999.
- [17] Zixin Wu, Karthik Gomadam, Ajith Ranabahu, Amit P. Sheth, and John A. Miller. Automatic Composition of Semantic Web Services Using Process Mediation. In *ICEIS 2007 - Proceedings of the Ninth International Conference on Enterprise Information Systems*, pages 453–462, 2007.