Open Geospatial Consortium, Inc.

Date: 2009-10-09

Reference number of this document: OGC 09-053r5

Version: 0.3.0

Category: Public Engineering Report

Editor(s): Bastian Schäffer

# OGC® OWS-6 Geoprocessing Workflow Architecture Engineering Report



**Warning**

This document is not an OGC Standard. This document is an OGC Public Engineering Report created as a deliverable in an OGC Interoperability Initiative and is not an official position of the OGC membership. It is distributed for review and comment. It is subject to change without notice and may not be referred to as an OGC Standard. Further, any OGC Engineering Report should not be referenced as required or mandatory technology in procurements.

| | |
|---|---|
| Document type: | OpenGIS® Engineering Report |
| Document subtype: | NA |
| Document stage: | Approved for Public Release |
| Document language: | English |

## Preface

This document presents the Geoprocessing Workflows related results of the OWS 6 GPW thread. This group has focused on creating general recommendations and guidelines for asynchronous workflows and security related aspects in workflows.

*Attention is drawn to the possibility that some of the elements of this document may be the subject of patent rights. The Open Geospatial Consortium Inc. shall not be held responsible for identifying any or all such patent rights.*

*Recipients of this document are requested to submit, with their comments, notification of any relevant patent claims or other intellectual property rights of which they may be aware that might be infringed by any implementation of the standard set forth in this document, and to provide supporting documentation.*

## License Agreement

Permission is hereby granted by the Open Geospatial Consortium, ("Licensor"), free of charge and subject to the terms set forth below, to any person obtaining a copy of this Intellectual Property and any associated documentation, to deal in the Intellectual Property without restriction (except as set forth below), including without limitation the rights to implement, use, copy, modify, merge, publish, distribute, and/or sublicense copies of the Intellectual Property, and to permit persons to whom the Intellectual Property is furnished to do so, provided that all copyright notices on the intellectual property are retained intact and that each person to whom the Intellectual Property is furnished agrees to the terms of this Agreement.

If you modify the Intellectual Property, all copies of the modified Intellectual Property must include, in addition to the above copyright notice, a notice that the Intellectual Property includes modifications that have not been approved or adopted by LICENSOR.

THIS LICENSE IS A COPYRIGHT LICENSE ONLY, AND DOES NOT CONVEY ANY RIGHTS UNDER ANY PATENTS THAT MAY BE IN FORCE ANYWHERE IN THE WORLD.

THE INTELLECTUAL PROPERTY IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, AND NONINFRINGEMENT OF THIRD PARTY RIGHTS. THE COPYRIGHT HOLDER OR HOLDERS INCLUDED IN THIS NOTICE DO NOT WARRANT THAT THE FUNCTIONS CONTAINED IN THE INTELLECTUAL PROPERTY WILL MEET YOUR REQUIREMENTS OR THAT THE OPERATION OF THE INTELLECTUAL PROPERTY WILL BE UNINTERRUPTED OR ERROR FREE. ANY USE OF THE INTELLECTUAL PROPERTY SHALL BE MADE ENTIRELY AT THE USER'S OWN RISK. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR ANY CONTRIBUTOR OF INTELLECTUAL PROPERTY RIGHTS TO THE INTELLECTUAL PROPERTY BE LIABLE FOR ANY CLAIM, OR ANY DIRECT, SPECIAL, INDIRECT OR CONSEQUENTIAL DAMAGES, OR ANY DAMAGES WHATSOEVER RESULTING FROM ANY ALLEGED INFRINGEMENT OR ANY LOSS OF USE, DATA OR PROFITS, WHETHER IN AN ACTION OF CONTRACT, NEGLIGENCE OR UNDER ANY OTHER LEGAL THEORY, ARISING OUT OF OR IN CONNECTION WITH THE IMPLEMENTATION, USE, COMMERCIALIZATION OR PERFORMANCE OF THIS INTELLECTUAL PROPERTY.

This license is effective until terminated. You may terminate it at any time by destroying the Intellectual Property together with all copies in any form. The license will also terminate if you fail to comply with any term or condition of this Agreement. Except as provided in the following sentence, no such termination of this license shall require the termination of any third party end-user sublicense to the Intellectual Property which is in force as of the date of notice of such termination. In addition, should the Intellectual Property, or the operation of the Intellectual Property, infringe, or in LICENSOR's sole opinion be likely to infringe, any patent, copyright, trademark or other right of a third party, you agree that LICENSOR, in its sole discretion, may terminate this license without any compensation or liability to you, your licensees or any other party. You agree upon termination of any kind to destroy or cause to be destroyed the Intellectual Property together with all copies in any form, whether held by you or by any third party.

Except as contained in this notice, the name of LICENSOR or of any other holder of a copyright in all or part of the Intellectual Property shall not be used in advertising or otherwise to promote the sale, use or other dealings in this Intellectual Property without prior written authorization of LICENSOR or such copyright holder. LICENSOR is and shall at all times be the sole entity that may authorize you or any third party to use certification marks, trademarks or other special designations to indicate compliance with any LICENSOR standards or specifications.

This Agreement is governed by the laws of the Commonwealth of Massachusetts. The application to this Agreement of the United Nations Convention on Contracts for the International Sale of Goods is hereby expressly excluded. In the event any provision of this Agreement shall be deemed unenforceable, void or invalid, such provision shall be modified so as to make it valid and enforceable, and as so modified the entire Agreement shall remain in full force and effect. No decision, action or inaction by LICENSOR shall be construed to be a waiver of any rights or remedies available to it.

None of the Intellectual Property or underlying information or technology may be downloaded or otherwise exported or reexported in violation of U.S. export laws and regulations. In addition, you are responsible for complying with any local laws in your jurisdiction which may impact your right to import, export or use the Intellectual Property, and you represent that you have complied with any regulations or registration procedures required by applicable law to make this license enforceable

## OWS-6 Testbed

OWS testbeds are part of OGC's Interoperability Program, a global, hands-on and collaborative prototyping program designed to rapidly develop, test and deliver Engineering Reports and Change Requests into the OGC Specification Program, where they are formalized for public release. In OGC's Interoperability Initiatives, international teams of technology providers work together to solve specific geoprocessing interoperability problems posed by the Initiative's sponsoring organizations. OGC Interoperability Initiatives include test beds, pilot projects, interoperability experiments and interoperability support services - all designed to encourage rapid development, testing, validation and adoption of OGC standards.

In April 2008, the OGC issued a call for sponsors for an OGC Web Services, Phase 6 (OWS-6) Testbed activity. The activity completed in June 2009. There is a series of on-line demonstrations available here: http://www.opengeospatial.org/pub/www/ows6/index.html The OWS-6 sponsors are organizations seeking open standards for their interoperability requirements. After analyzing their requirements, the OGC Interoperability Team recommended to the sponsors that the content of the OWS-6 initiative be organized around the following threads:

1. Sensor Web Enablement (SWE)
2. Geo Processing Workflow (GPW)
3. Aeronautical Information Management (AIM)
4. Decision Support Services (DSS)
5. Compliance Testing (CITE)

The OWS-6 sponsoring organizations were:

- U.S. National Geospatial-Intelligence Agency (NGA)
- Joint Program Executive Office for Chemical and Biological Defense (JPEO-CBD)
- GeoConnections - Natural Resources Canada
- U.S. Federal Aviation Agency (FAA)
- EUROCONTROL
- EADS Defence and Communications Systems
- US Geological Survey
- Lockheed Martin

- BAE Systems
- ERDAS, Inc.

The OWS-6 participating organizations were:
52North, AM Consult, Carbon Project, Charles Roswell, Compusult, con terra, CubeWerx, ESRI, FedEx, Galdos, Geomatys, GIS.FCU, Taiwan, GMU CSISS, Hitachi Ltd., Hitachi Advanced Systems Corp, Hitachi Software Engineering Co., Ltd., iGSI, GmbH, interactive instruments, lat/lon, GmbH, LISAsoft, Luciad, Lufthansa, NOAA MDL, Northrop Grumman TASC, OSS Nokalva, PCAvionics, Snowflake, Spot Image/ESA/Spacebel, STFC, UK, UAB CREAF, Univ Bonn Karto, Univ Bonn IGG, Univ Bunderswehr, Univ Muenster IfGI, Vightel, Yumetech

# Contents

Page

# Figures

Page

# Tables

Page



# Listings

# OGC® OWS-6 Geoprocessing Workflow Engineering Report

## 1 Introduction

### 1.1 Scope

This document covers Geoprocessing Workflow best practices and methods in a SOA environment. A RESTful approach was also conducted in this testbed, but no specific implementation details were available to be included in this ER; also, the RESTful workflow approaches and technology used in this testbed was essentially same as that used in OWS-5.

### 1.2 Document contributor contact points

All questions regarding this document should be directed to the editor or the contributors:

| Name | Organization |
|---|---|
| Bastian Schaeffer | Institute for Geoinformatics, University of Muenster |
| Sven Schade | Institute for Geoinformatics, University of Muenster |
| | |

### 1.3 Revision history

| Date | Release | Editor | Primary clauses modified | Description |
|---|---|---|---|---|
| 17.4.09 | 0.9 | Schäffer, Schade | | Completed draft |
| 18.4.09 | 0.9.1 | Schäffer | 5.5 | Typos eliminated, BPEL scripts for asynchronous support added, references added |
| 20.4.09 | 0.9.2 | Schade, Schäffer | | Figure 9 Added, typos eliminates |
| 25.4.09 | 0.9.3 | Schäffer | | References updated and extended |
| 22.7.09 | 0.3.0 | Schäffer | | Polishing of all sections; indication that RESTful approaches are not part of the ER. |
| 08/10/09 | 0.3.0 | C. Reed | | Get ready for posting a public ER |

### 1.4 Future work

## 2 References

## 3 Conventions

### 3.1 Abbreviated terms

| | |
|---|---|
| ASM | Abstract State Machines |
| BPEL | Business Process Execution Language |
| GIS | Geographic Information System |
| GML | Geographic Markup Language |
| GPW | Geo-Processing Workflow |
| ISO | International Organization for Standardization |
| KVP | Key-Value Pair |
| OGC | Open Geospatial Consortium |
| QoS | Quality of Service |
| SOA | Service Oriented Architecture |
| SWS | Semantic Web Service |
| UDDI | Universal Description, Discovery and Integration |
| UML | Unified Modeling Language |
| W3C | World Wide Web Consortium |
| WCS | Web Coverage Service |
| WFS | Web Feature Service |
| WMS | Web Map Service |
| WPS | Web Processing Service |
| WSDL | Wed Service Description Language |
| WSML | Web Service Modelling Language |

WSMO Web Service Modelling Ontology

WSMX Web Service Modelling eXecution engine

XML eXtensible Markup Language

### 3.2 UML notation

Most diagrams that appear in this standard are presented using the Unified Modeling Language (UML) static structure diagram, as described in Subclause 5.2 of (OGC 2007).

## 4 Introduction

This section starts with an overview over the OGC activities and achievements in the context of Geoprocessing Workflows in the past. This serves as a starting point for defining the term *Geoprocessing Workflow* followed by explanations of relevant concepts for challenges targeted at the OWS-6 GPW testbed. The presented concepts are evaluated with a proof-of-concept implementation at the end of this document.

### 4.1 Basic concepts for OWS Workflows

The Open Geospatial Consortium has focused on spatial related workflows since several years. Starting with ISO19119 (ISO 2001) the OpenGIS Consortium (OGC) and ISO TC211 have jointly developed an international standard for geospatial service architecture including the description of different workflow patterns (see section 4.2).

Additionally, several testbeds explored Geoprocessing Workflows in detail: In the OWS-2 testbed, service chaining with the Business Process Execution Language (BPEL) was elaborated. Figure 1 shows the basic architecture.

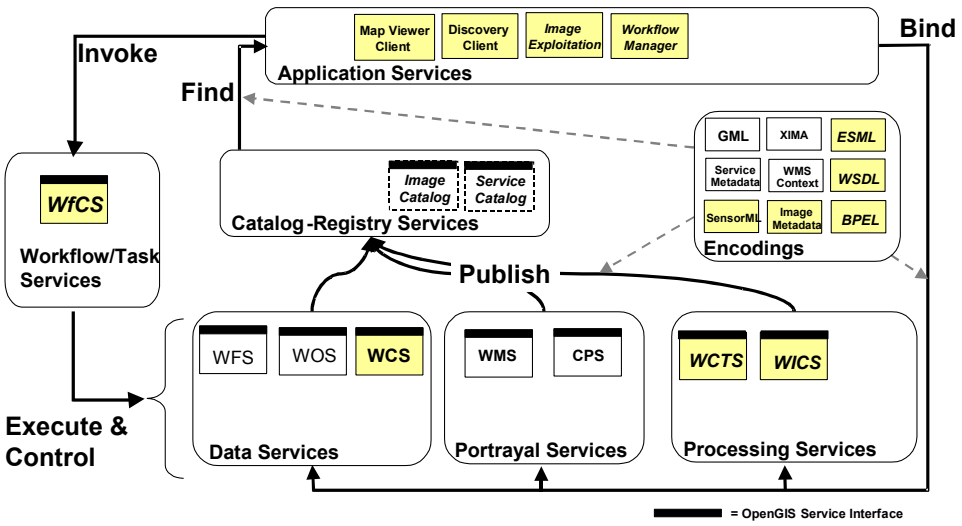



**Figure 1. OWS-2 Architecture**

Details are described in (OGC 2004) but it is important to note that a BPEL workflow engine was used and labeled as a Workflow Chaining Service (WfCS). Therefore a WfCS is a class of Workflow and Task Services as defined in ISO 19119 but is not defined further with a fixed interface and therefore can be only regarded as a concept rather than a service in an OGC sense. Since the workflow is hidden behind the vendor specific WfCS interface, the opaque service chaining pattern was applied. Additionally, (OGC 2004) describes only synchronous workflow interactions but identified a need for asynchronous service interaction as well. All data was passed by reference since all services (processing and data) supported this type of data transaction.

The follow-up OWS-3 testbed also relied on BPEL as the workflow language. As shown in figure 2, a workflow consisting of Web Coordinate Transformation Service (WCTS) was implemented for a remote sensing scenario.

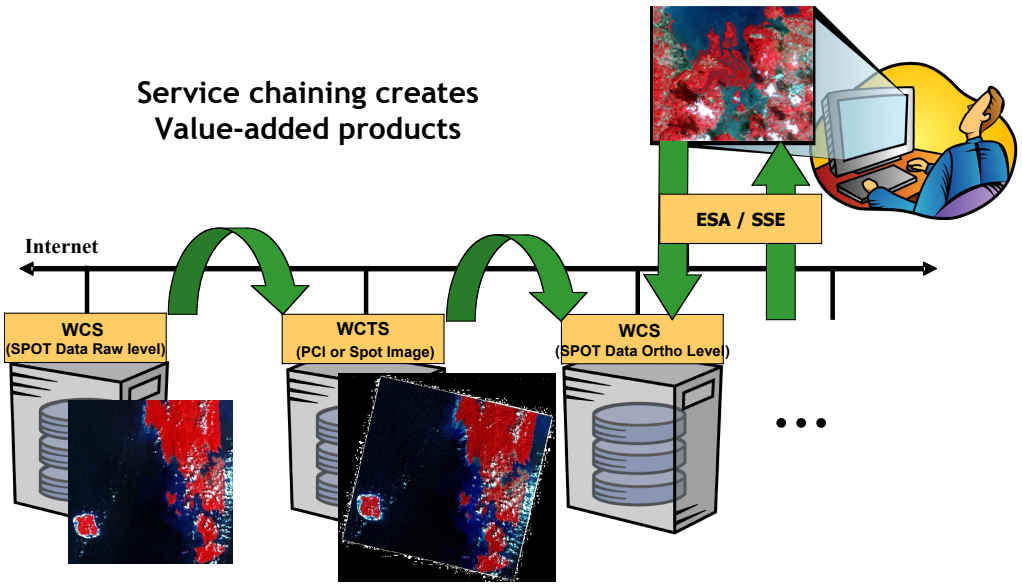



**Figure 2. OWS-3 Service Chain**

Again the vendor specific interface of the Workflow Engine was directly used and referred to as WfCS. Also, asynchronous service invocation with WS-Addressing was elaborated and implemented for the WCTS.

The next evolution step took place in the OWS-4 testbed. Again, several workflows had to be implemented and BPEL was selected again as the workflow language and the workflow was hidden behind a standardized WFS/SPS interface. Therefore, the WFS-T/SPS operations like GetFeature triggered the workflow on a BPEL workflow engine. Figure 3 shows the basic architecture for one of the workflows developed.

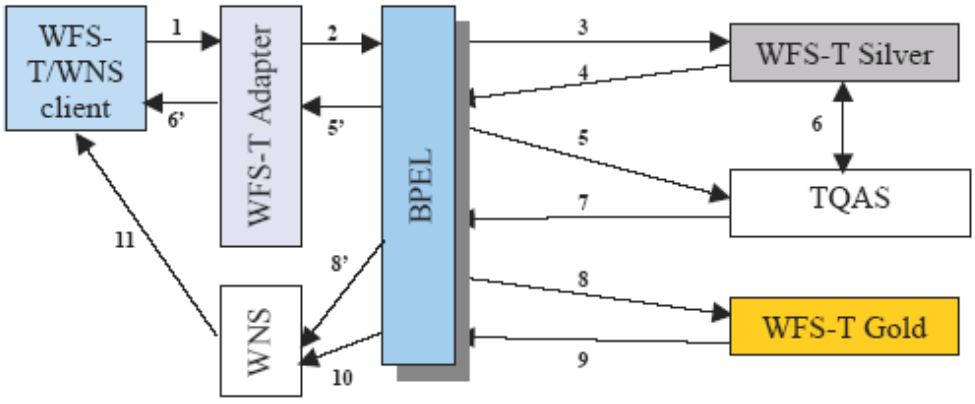



**Figure 3. OWS-4 Workflow Architecture**

Details are described in (OGC 2006). However, asynchronous invocation of the service was possible with the help of the Web Notification Service (WNS) (OGC 2003). The workflow itself was still synchronous.

In the OWS-5 testbed, two completely different workflow approaches were exercised. The first one applied the SOA principle and continued the practice of synchronous BPEL scripts but wrapped the workflow as an asynchronous WPS process which delivers a reference to a WFS back as show in Figure 4. Besides, the workflow was based on the SOAP protocol.

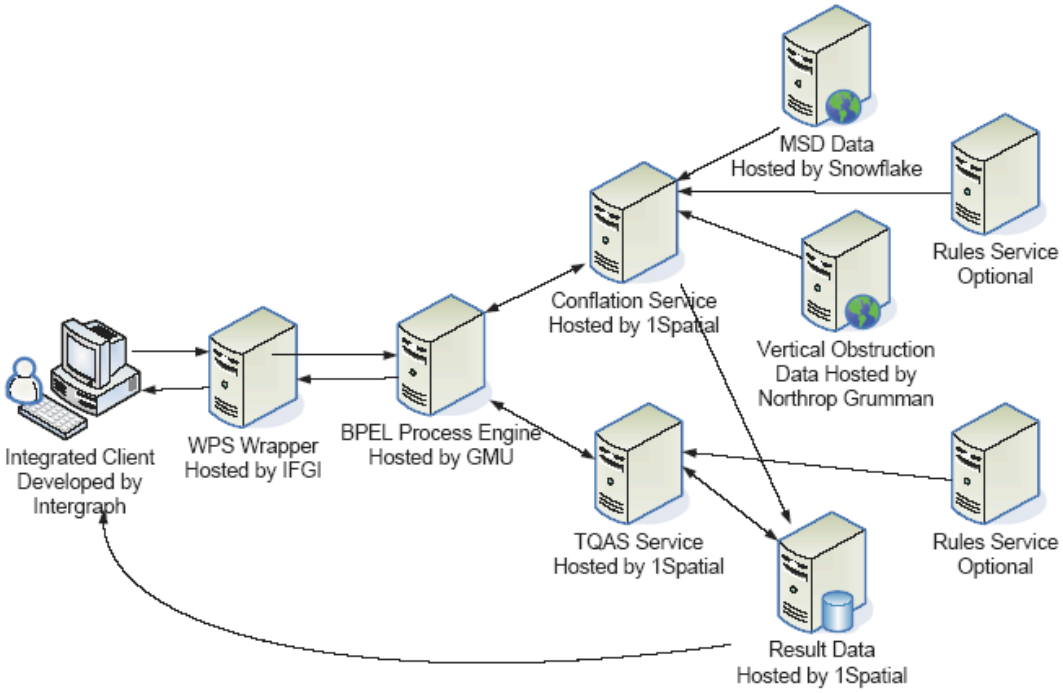



**Figure 4. OWS-5 SOA Workflow Architecture**

Beyond the OWS-5 context, the OGC discussion paper (OGC 2008) extended the approach of wrapping workflows as WPS processes to a dynamic deployment/undeployment of any algorithm/workflow as a WPS process. As a result, the WPS interface was extended to a Transactional WPS (WPS-T).

The second OWS-5 workflow applied the ROA principle and wrapped different workflow engines behind a RESTful version of WF-XML (WF-XML-R) interface as shown in figure 5.

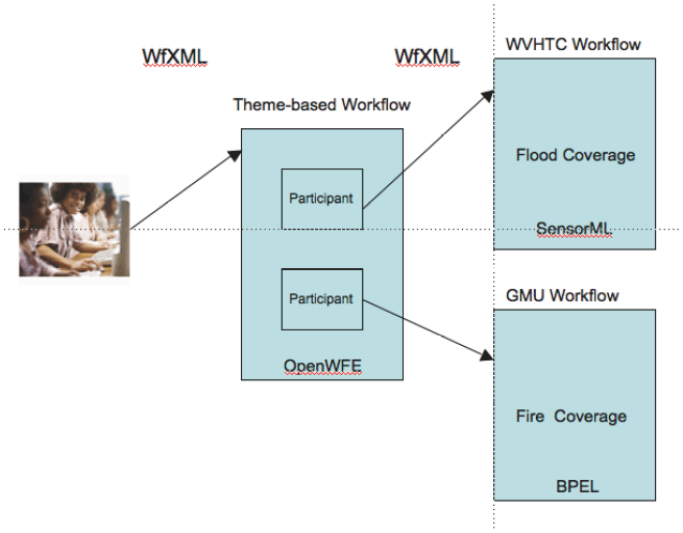



**Figure 5.** OWS-5 ROA Workflow Architecture

Besides, limited security capabilities for RESTful workflow were apparently exercised based on an OpenID extension but not explained further in the OWS-5 Workflow Engineering report.

### 4.2 Geoprocessing Workflow

This section provides a brief classification of the term *Geoprocessing Workflow* as the key aspect of this report.

*Geoprocessing Workflows* can be viewed as a combination of the two general concepts *Geoprocessing* and *Workflow*.

**Geoprocessing**

In absence of a general definition for the term *Geoprocessing*, it can be seen as a specialization of the term processing in a spatial context. In other words, *Geoprocessing* is the processing of spatially related data.

In classical desktop GIS applications, geoprocessing represents the core GIS analysis functionality (Lembo 2004) and thus, is one of the key concepts. On the other side, distributed GIS systems are based on loosely coupled services organized in a SDI (Kiehle et al. 2006). According to the ISO 19119 specification (ISO 2001), there are four different types of Geoprocessing services:

- Spatial processing

e.g. Coordinate transformation services, Feature manipulation services or Route determination services

- Thematic processing

e.g. Thematic classification services, Geographic information extraction services, or Image processing services

- Temporal processing

e.g. Temporal reference system transformation services, Subsetting services, Temporal proximity analysis services

- Metadata processing

e.g. Statistical calculation service, Geographic annotation services

**Workflow**

The term *Workflow* is defined as an "automation of a business process, in whole or part, during which documents, information or tasks are passed from one participant to another for action, according to a set of procedural rules" (ISO, 2001).

Therefore, the concept of a workflow can be realized as a web service chain in order to pass information from one workflow participant (web service) to another.

**Geoprocessing Workflows**

*Geoprocessing Workflow* brings both terms together. In the scope of this report, it can be seen as an automation of a spatial process/model, in whole or part, during which information is passed from one distributed Geoprocessing Service to another according to a set of procedural rules using standardized interfaces.

In other words, Geoprocessing Workflows integrate data and services in an interoperable way, where each part of the workflow is responsible for only a specific task, without being aware of the general purpose of the workflow. Due to the distributed nature of geographic data, Geoprocessing Workflows provide flexible means of processing highly distributed and complex data for a wide variety of uses.

Leaving the semantical aspect out of scope, some research has recently been accomplished on web based geoprocessing. Kiehle (Kiehle et al., 2006) describes a rule based approach and theoretical foundations for geoprocessing service orchestration. Weiser (Weiser et al. 2006) and Stollberg (Stollberg, 2006), focus on BPEL and WPS but do not take the whole Geoprocessing lifecycle and OGC constraints into account. But the OGC Interoperability and Specification Programs has produced a significant body of knowledge and experience in designing, building and deploying Web Services. The full potential of OGC Web Services as an integration platform will be achieved when applications and business processes can be composed to perform complex interactions using a standardized process integration approach.

Workflows regarded as Service Chains are one of the key concepts in enabling value-added chains in SDIs (Alameh 2003). The ISO19119/Service Architecture standard defines service chaining as: “A sequence of services where, for each adjacent pair of services, occurrence of the first action is necessary for the occurrence of the second action“ (ISO, 2001).

A service chain can always be regarded as a directed graph, since the input of one service depends on the output of another service. A directed graph is defined as:

A set *K* of ordered nodes and a set *E* of edges, where each edge *e(u,v)ЄE* has a direction and consists of a node pair *(u,v)* where *u,v ЄK*. .

The nodes in a directed graph represent service entities and the arcs represent the service interactions. Directed acyclic graphs (DAG) are special types of directed graphs. The definition of a directed graph from above has to be extended with the constraint that for any node *t*, there is no nonempty directed path that starts and ends on *t*.

However, some service chains require iterations and for this reason the graph has to be cyclic and therefore has to make use of conditions in the control function to address convergence.

In addition, there are four more characteristics of a service chain according to ISO19119 (ISO 2001):

- Parallel or serial chains
- Variations in the links between nodes reflecting different methods for transporting data or invoking the service
- Parameters in nodes
- pull processing vs. push processing

Besides, there are three different architectural patterns for service chains defined according to Alameh (Alameh 2003) and ISO19119 (ISO, 2001): User Transparent chaining, translucent chaining and opaque chaining.

### 4.2.1 Transparent Chaining

In the transparent chaining pattern, the knowledgeable user defines a service chain. Since all service details are visible to the user, this pattern is called transparent chaining. Thus, the user is responsible for discovering and evaluating available services as well as for defining the execution order, invoking the services and pass around process results as inputs. Furthermore, the user has to make sure that input and output messages have to be compatible and all required resources are available. Figure 6 presents this pattern as an UML collaboration diagram.

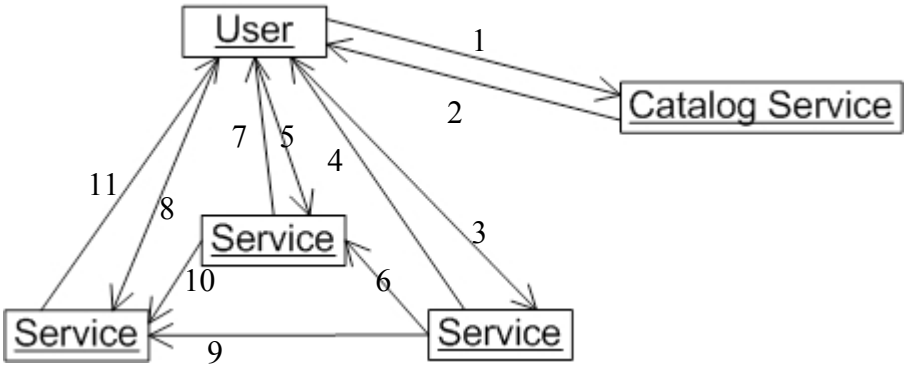



**Figure 6.** Transparent Chaining Pattern

In step 1, the user sends a search request to a catalog service in order to discover service availability. The catalog service returns metadata about services fulfilling the search request. The user creates an execution order and invokes the first service in step 3. The processed results (or a reference) are returned back to the user in step 4. These results (or a reference) are passed in the second service in step 5. If the user supplies a reference, the service has to obtain the actual data in step 6 from the previous service. Again, the processed results (or a reference) are returned back to the user as can be seen in step 7. In step 8, the user invokes the third service with the results from the two previous services. If a reference to actual data is delivered, the third service requests the actual data from the corresponding in step 9 and 10. After processing, the final results are returned to the user in step 11.

For instance, typical applications of transparent chaining are Web 2.0 style mash-ups. These mash-ups allow e.g. adding a geotagged pictures to a map.

### 4.2.2 Translucent Chaining

The translucent chaining pattern allows a user to execute a predefined chain managed by a workflow service. In this pattern, the chain is already abstractly predefined and stored on a workflow engine. The user is aware of all services participating in the chain, but does not have to deal with the execution order or passing around processing results. But since the user knows all participating services, he is able to poll the current status of each participating service (if supported by the service). Figure 7 gives an overview of this pattern.

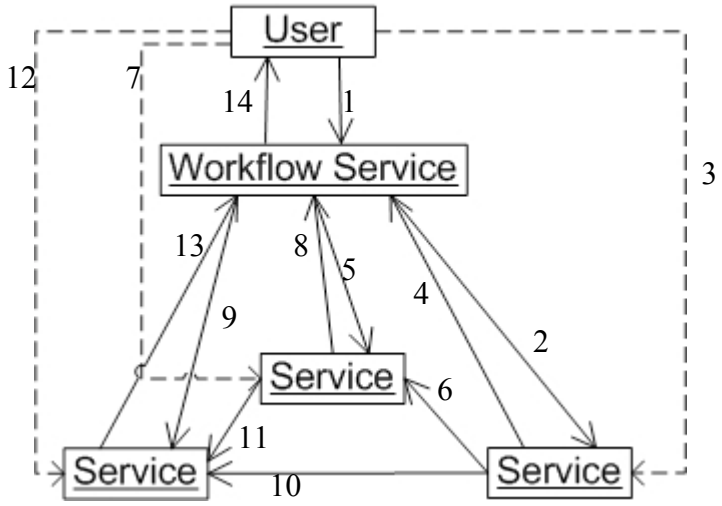



**Figure 7.** Translucent Chaining Pattern

In step 1, the user invokes an existing chain on the workflow service. The workflow service starts now the predefined execution order. The first service is invoked in step 2. Since the user knows, that this service is invoked first, he/she can poll the current processing status of the service. The processed results (or a reference) are returned back to the workflow service in step 4. These results (or a reference) are passed in the second service in step 5. If the workflow service supplies a reference, the service has to obtain the actual data in step 6 from the previous service. Again, the user can poll the status in step 7 and after processing, the results (or a reference) are returned back to the workflow service as can be seen in step 8. In step 9, the workflow service invokes the third service with the results from the two previous services. If a reference to actual data is delivered, the third service requests the actual data from the corresponding in step 10 and 11. As seen before, the user is enabled to poll the current service status in step 12. After processing, the final results are returned to the workflow service in step 13 and from there back to the user in step 14.

### 4.2.3 Opaque Chaining

The opaque chaining pattern exposes a chain as a single service and hides all details from the user. The user is not even aware of the fact that the aggregate service hides a chain nor is the user aware of the types of services being used. Therefore, the aggregate service is responsible for all service coordination. Figure 8 describes this pattern.

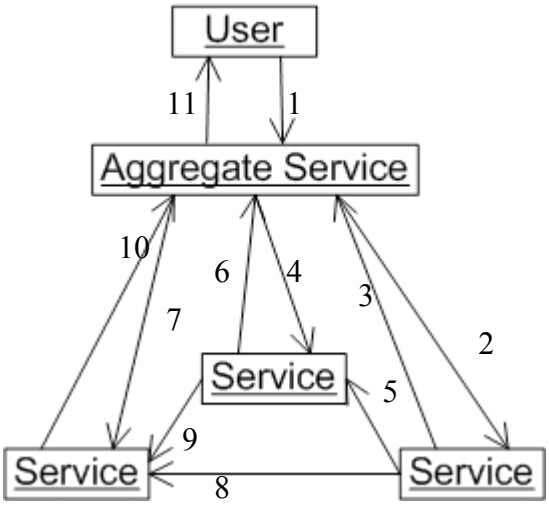



**Figure 8.** Opaque Chaining Pattern

In step 1, the user invokes an aggregate service unaware of the implementation details (e.g. if the aggregate service uses a service chain and what kind of services are involved). The aggregate service starts now the predefined execution order at this point, so the first service is invoked in step 2. The processed results (or a reference) are returned back to the aggregate engine in step 3. These results (or a reference) are passed in the second service in step 4. If the aggregate service supplies a reference, the service has to obtain the actual data in step 5 from the previous service. Again, after processing, the results (or a reference) are returned back to the aggregate service as can be seen in step 6. In step 7, the aggregate service invokes the third service with the results from the two previous services. In case a reference to actual data is delivered, the third service requests the actual data from the corresponding service as presented in step 8 and 9. After processing, the final results are returned to the aggregate service in step 10 and from there back to the user in step 11.

#### 4.2.4 Business Process Execution Language (BPEL)

The de-facto standard for Web Service workflow composition (van der Aalst et al., 2003a) is the Business Process Execution Language for Web Services (BPEL4WS, commonly referred as BPEL), which is a standard proposed by IBM, Microsoft and BEA (Andrews et al., 2003a), that evolved from former standards, such as graph based WSFL or block based and algebraic XLANG. BPEL descriptions (scripts) are XML based documents, which describe the roles involved in the message exchange, supported port types and orchestration information of a process. BPEL is also a service composition model (Wohed et al., 2003), which supports composition and coordination protocols (Chen et al., 2006). It consists of an activity-based component model, an orchestration model that allows the definition of structured activities, XML schema data types, a service selection model and a mechanism for exception and event handling.

A BPEL process does not only interact with a set of partners through Web Service interfaces but also represents a Web Service from an external point of view (figure 9). This process is internally implemented by the interaction of participating Web Services, whose interfaces are specified by WSDL documents. Reuse and simplicity of BPEL processes are guaranteed by operating only on the "abstract" PortType definitions instead of the concrete Port definitions of a Web Service. Thereby, specific binding and deployment issues remain at the Web Service and not at the BPEL process itself (Leymann et al., 2003).

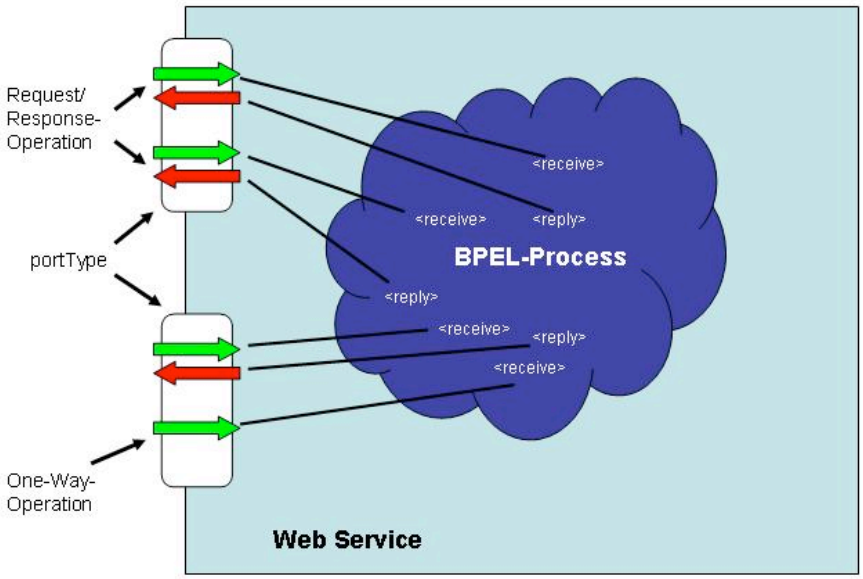



**Figure 9.** BPEL Process Overview

Since BPEL is strongly related to Web Services, BPEL4WS is build on top of numerous XML based specifications: WSDL 1.1, XML Schema 1.0 and XPath 1.0. WSDL describes all external resources and participating Web Services. XML Schema specifies the datatypes in conjunction with the WSDL messages and XPath is needed for internal data manipulation.

#### 4.2.4.1 BPEL process lifecycle

BPEL is built on WSDL by assuming that all external interactions of the business process occur through Web Service operations. However, a BPEL process is only a model that represents a stateful long-running interaction in which each interaction has a beginning, a defined behaviour during its lifetime, and an end. Thus, BPEL processes are instantiated at run-time and not a design time.

Creating a BPEL process instance is always implicit in the way that activities which receive external messages (`<receive>`) can be annotated to indicate that a new instance of the business process has to be created. This is achieved by setting the `createInstance` attribute to "`yes`". When a message is received, an instance of the process is created if it does not already exist.

A BPEL process instance is terminated if either all activities that characterize the behaviour of a process are completed (normal termination) or if an error occurs. Occurring errors can be either handled or not but. In either case, a response is sent to the requestor.

#### 4.2.4.2 BPEL process general structure

A BPEL process follows a general structure presented in listing 1

```
1  <process name="processname">
2      <partnerLinks> ... </partnerLinks>
3      <partners> ... </partners>
4      <variables> ... </variables>
5      <correlationSets> ... </correlationSets>
6      <faultHandlers> ... </faultHandlers>
7      <compensationHandler> ... </compensationHandler>
8      <eventHandlers> ... </eventHandlers>
9
10     activity
11
12 </process>
```

**Listing 1.** General Structure of a BPEL process

A `<process>` root element is equipped with a mandatory name attribute representing the name of the process. All listed child elements are optional but at least one activity has to be provided. This activity can be either a basic activity like invoking a Web Service operation or a control structure activity. The following Sections give a brief overview of the elements.

#### 4.2.4.3 Partner Handling

The most important aspect of a BPEL process is the interaction with other Web Services (`partner`). The relationship of a process to a partner is typically peer-to-peer. This aspect requires a two-way dependency at the service level. Therefore, a partner can represent both a consumer of a service provided by the process and a provider of a service to the process. The interaction is generally handled through operation calls offered by the Web Service's interface and described by the corresponding WSDL. WSDL describes the Web Service's functionality only and does not include the relationships between different partners. Therefore BPEL offers the concept of `<partnerLinks>` and `<partnerLinkTypes>`.

A `<partnerLinkType>` determines the relationship between two services and also characterizes the roles of both partners. Additionally, the interaction pattern is specified between two partners by defining the `<portType>` provided by each service to receive messages within the context of the conversation. Hence, a `<partnerLink>` specifies only the interaction at an abstract level by simply defining just the `<roles>` and `<portTypes>` and not the concrete `ports`. Listing 2 shows the general structure of a `<partnerLinkType>`.

```
1  <partnerLinkType name="name">
2    <role name="name">
3      <portType name="qname"/>
4    </role>
5    <role name="name">
6      <portType name="qname"/>
7    </role>
8  </partnerLinkType>
```

**Listing 2.** General partnerLinkType structure

Each `<partnerLinkType>` construct is named after the name attribute. Furthermore, the Web Service's role (e.g. buyer and seller) is specified by the role element, which is also named. Each role specifies exactly one `<portType>` provided for the corresponding role. A `<partnerLinkType>` is different from other BPEL constructs, since it is specified in the context of a WSDL document on the basis of the WSDL extension mechanism and hence can be placed inside a WSDL document.

All Web Services, which participate in the process and therefore interact with the process are specified as `<partnerLinks>`. Each `<partnerLink>` is characterized by a `<partnerLinkType>`. But more than one `<partnerLink>` can be characterized by

the same <partnerLinkType>. Listing 3 shows the general structure of a <partnerLink>.

```
<partnerLinks>
  <partnerLink name="ncname" partnerLinkType="qname"
  myRole="ncname"? partnerRole="ncname"?></partnerLink>
</partnerLinks>
```

**Listing 3.** Sample PartnerLink structure

Each <partnerLink> has a name attribute and references the corresponding <partnerLinkType> by the partnerLinkType attribute. The role of the process is represented by the attribute myRole and the role of a Web Service represented by this <partnerLink> is indicated by the attribute partnerRole.

#### 4.2.4.4 Data Handling

BPEL processes are usually long-time running and time consuming. For the interaction of the process with its partners, BPEL offers the concept of variables to temporally store data. Variables can be allocated with received data from a partner and act as input data for another partner. The BPEL specification defines several constructs to handle, extract and manipulate data and variables. The most important elements are described below.

Variables serve in the context of BPEL to hold data received or send to partners. Hence, they represent the state of the BPEL process. Due to BPEL's strong type-safety, variables have to be declared before being used. The datatype of variables can be determined by either WSDL messages, complex XML schema elements or simple XML schema elements, e.g. xsd:int. Listing 4 presents a sample variable declaration.

```
<variables>
  <variable name="name" messageType="qname"
  type="qname" element="qname"/>
</variables>
```

**Listing 4.** Sample Variable Declaration

Inside the enclosing <variables> element, there are variables which can be defined. A <variable> has a name under which it is available for other constructs inside the BPEL process. One of the following elements has to be provided in order to specify the datatype: <messageType> for WSDL messages, <type> for simple XML types and <element> for XML schema elements.

Variables are only useful if they can be assigned with data and moreover the data can be copied or modified. BPEL offers the <assign> activity to copy data between variables. This activity is characterized by one or many <copy> activities, which copy data from one source (<from>) to a type compatible source (<to>). Listing 5 shows an example for copying data between WSDL messages.

```
1  <assign>
2    <copy>
3      <from variable ="name" part="name" query="queryString"/>
4      <to variable="name" part="name" query="queryString"/>
5    </copy>
6  </assign>
```

**Listing 5.** Sample assign activity

The source variable is determined by its name. Since listing 5 shows the syntax for copying data between WSDL messages, the part of the WSDL message is obtained by the part attribute and an optional XPath query can be executed on the part specified by the query attribute. The target variable mechanism is analogous to the source variable behaviour.

#### 4.2.4.5 Basic Activities

Basic activities are the building blocks of the actual process workflow. Only the major activities <invoke>, <receive> and <reply> are briefly described in this Section. Besides, there are other activities as mentioned in previous sections, such as the <assign> activity and the <throw>, <wait> and <empty> activity which are not covered here.

The <invoke> activity characterizes an operation call on an interface provided by a web service and specified by the <partnerLink> element. Input and output data has to be copied from or to BPEL variables, which are typed as WSDL message. Synchronous request/response operations and asynchronous one-way operations are supported by the BPEL specification. Listing 6 shows a typical <invoke> activity.

```
1  <invoke inputVariable="name" operation="operation"
2  outputVariable="name" partnerLink="name" portType="name"
3  />
```

**Listing 6.** Sample invoke element structure

The `inputVariable` and the `outputVariable` have to be specified by their `name`. The `operation` attribute characterizes the operation to be called on the interface identified by the `portType` attribute and at the partner specified by the `partnerLink` attribute.

From an external point of view, every BPEL process is represented as a simple Web Service, the required input data and the generated output data have to be transformed as served/requested by the interface operations. The `<receive>` activity is triggered when an associated operation is called. This activity writes the incoming data into a dedicated variable for further use. If a process is not a one-way process, a `<reply>` activity performs the opposite of the `<receive>` operation. The `<reply>` activity returns variable data back to a waiting requestor[1]. Listing 7 presents both activities.

```
1  <receive variable="name" operation="operation"
2   createInstance="yes" partnerLink="name" portType="name"
3   />
4    .
5    .
6    .
7  <reply variable="name" operation="operation"
8   partnerLink="name" portType="name"
9   />
```

**Listing 7.** Principle Receive-Reply structure

The `<receive>` activity fetches the incoming data and assigns them to the variable indicated by the variable attribute. In case of the `<reply>` activity, the `variable` attribute represents the variable, which value should be returned to the requestor. In both cases, the `operation` attribute characterizes the operation to be used for fetching/returning the data of the interface identified by the `portType` attribute and the partner specified by the `partnerLink` attribute.

Besides these basic activities, the BPEL specification offers a set of control structure activities as known from high level programming languages (e.g. switch, while etc.) for controlling and structuring the sequence of activities. In the scope of this report, the `<sequence>` control structure activity is only relevant. The `<sequence>` control structure activity orders a containing set of activities in a sequential order. This becomes necessary for synchronous processes while asynchronous processes rely on the `<flow>` control structure activity.

[1] For synchronous calls

### 4.2.5 XPDL

XPDL provides an XML file format that can be used to interchange process models between tools. This specification forms part of the documentation relating to Interface 1 - supporting Process Definition Import and Export shown in the diagram above. This interface includes a common meta-model for describing the process definition (this specification) and also a companion XML schema for the interchange of process definitions.

An XPDL package corresponds to a Business Process Diagram (BPD) in Business Process Markup Notation (BPMN), and consists of a set of Process Definitions. The WfMC defines a process as:

> *The representation of a business process in a form that supports automated manipulation, such as modeling, or enactment by a workflow [or business] management system. The process definition consists of a network of activities and their relationships, criteria to indicate the start and termination of the process, and information about the individual activities, such as participants, associated IT applications and data, etc. (WfMC Glossary - WfMC-TC-1011)*

The process definition provides an environment for a rich description of a process that can be used for the following:

- Act as a template for the creation and control of instances of that process during process enactment.
- For simulation and forecasting.
- As a basis to monitor and analyse enacted processes.
- For documentation, visualization, and knowledge management.

The process definition may contain references to subflows, separately defined, which make up part of the overall process definition.

XPDL as a workflow language was used in the Restful part of the OWS-6 GPW testbed but no specific implementation details were available to be included in this ER.

### 4.2.6 WSMO/WSML/WSMX

Workflow generation and execution is also an emerging topic within the Semantic Web community. Considering related technology in the context of Geospatial workflow is highly relevant. In particular, because issues of web service discovery and mediation can be addressed. For these reasons the Web Services Modeling Ontology (WSMO) has been considered for OWS-6 implementations. Due to the current status of the software and a lack in support for web service security, developments were not completed. As we see a

strong requirement for semantic technologies in near future, the core principles of WSMO are detailed in the following. This overview serves a connection to further work.

The WSMO is introduced as solution to semantic discovery and composition of Web Services. Opposed to other Semantic Web technology, WSMO provides an integrated conceptual model for Web Service discovery, composition, and execution. WSMO defines a conceptual SWS model. It defines four basic conceptual modelling elements (Scicluna et al., 2006):

*WSMO Ontologies* define a common agreed-upon terminology for an application domain. Following Guarino, ontology is defined as "an engineering artifact, constituted by a specific vocabulary used to describe a certain reality, plus a set of explicit assumptions regarding the intended meaning of the vocabulary words" (Guarino, 1998).

*WSMO Web Services* are descriptions that formalize various aspects of a Web Service. A service capability defines the Web Service functionality in terms of non-functional properties, pre-conditions and post-conditions. The choreography defines how a service interacts with its clients. It is similar to a WSDL end point, and represents the external visible behaviour of a service, including its communication structure and grounding. WSMO Choreography is based on Abstract State Machines (ASMs). The orchestration describes how the provided functionality is realized internally, i.e. it specifies the interaction with the composed Web Services. A WSMO Orchestration is state-based and describes the desired workflow together with control and data flow. The orchestration description consists of a vocabulary that denotes the information space from the perspective of the Web Service. It consists of guarded transition of the form: if <condition> then invoke <Goal, Web Service>. WSMO Web Service descriptions use the available WSMO Ontologies. Invoking a Web Service means direct execution.

*WSMO Goals* describe the user's desires with respect to the requested functionality. A Goal description consists of a requested capability and required choreography interfaces. WSMO Goals use WSMO Ontologies. Achieving a WSMO Goal means that the appropriate service has to be discovered, and executed by using choreography and orchestration.

*WSMO Mediators* are used to deal with syntactical, structural and semantic heterogeneities inherent in open and flexible architectures, such as the Service Oriented Architecture (SOA).These heterogeneities can be on mismatches between different used terminologies (data level), on communicative behavior between services (protocol level), and on the business process level. A WSMO Mediator connects elements and provides mediation facilities for resolving such mismatches. They act as connectors between WSMO components.

Semantic Web Service (SWS) frameworks such as WSMO allow attaching meaning to terminology used in service and data description by using ontologies. For instance, service inputs, outputs and operations can be clearly defined, and hence ambiguities and heterogeneities in terminologies are reduced (Fitzner and Hoffmann, 2007). In this way

WSMO addresses poorly defined semantics, shared syntax with different semantics, as well as, different syntax with shared semantics.

A comparison of WSMO and BPEL can be found in (Gone and Schade, 2008). Opposed to BPEL, a WSMO service composition does not have to exist at the time when a user poses the request i.e. formulates the WSMO Goal. Compositions may employ different services, which is more flexible than using pre-selected services. The concept of goals is used in WSMO. Goals allow the specification of processes and tasks, for which suitable Web Services can be detected dynamically at execution time. WSMO includes discovery mechanisms for identifying Web Services with suited semantics.

The reference implementation of a WSMO is the Web Service Execution Engine (WSMX). WSMX uses the Web Service Modeling Language (WSML) as its internal language. Conditions are WSML axioms, which describe the states for which the transition rule should fire. By serving a modeling ontology for Web Services, WSMX is a general conceptual framework which remains open for extensions. In the following we introduce the internal workings of WSMX and the required language elements.

#### 4.2.6.1 Overall Architecture

WSMX itself is based on SOA principles. It is divided into self-contained components which provide different functionality to the system. From a service point of view, the components act as middleware service providers. The communication between the components is event-based, using a publish-subscribe mechanism.

The central component in WSMX is the Core, which provides middleware framework functionality such as finding and loading components, handling the messaging between components, and defining paths of execution (“execution semantics”, see below). The Core also defines the types of components in the Integration API. In essence, the Core is responsible for all cross-cutting concerns regarding the other components (i.e. the Core realizes the vertical layer as described in (Hoffmann et al., 2008).

The other components are decoupled from the each other (incl. from the Core itself) by wrappers. This architecture enables to easily plug in additional components and to switch any of the provided components to a third-party component that realizes the same functionality.

The currently available components, apart from the Core, include: choreography, communication manager, data mediator, invoker, orchestration, parser, resource manager, service discovery, and web service discovery. The following section gives an example how they are used in WSMX to process SWS.

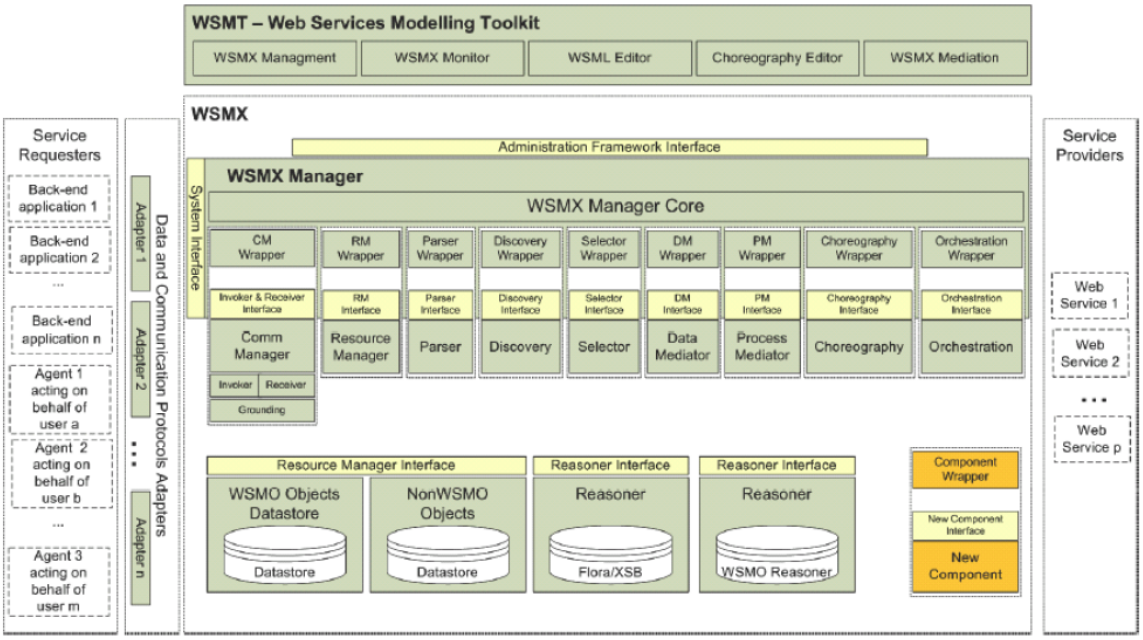



**Figure 10.** WSMX Architecture.

WSML (deBruijn et al., 2006) is a family of formal description languages used for the precise specification of the elements in the WSMO framework. The different variants of WSML (WSML-Core, WSML-Flight, WSML-Rule, WSMLDL, and WSML-Full) correspond to different logical language paradigms, namely Description Logic, Logic Programming and First-Order Logic.

#### 4.2.6.2 WSMX Execution Semantics - exemplified by AchieveGoal

Execution semantics can be viewed as executable processes that define a path through the system. Thus they enable the combined execution of functional components. Execution semantics are specified by the core and are triggered by specific requests to the system.

Currently, there is one execution semantic specified: AchieveGoal. It defines the goal-based invocation of services, i.e. the process to be executed in order to get a response for a specified goal.

AchieveGoal comprises of the following steps

1. **Late Binding: Service Discovery and Selection** (the following steps are executed sequentially and just once each):

    a. **Web Service Discovery**: The given goal definition is used to find web service definitions in the repository which can fulfil that goal.

b. **Service Discovery** (optional): It may be necessary that web services need to be invoked with the actual instance data that was provided with the goal in order to determine whether a service can actually fulfil the concrete goal2. (not used in swing)

c. **Service Selection** (optional): Additional criteria like Quality of Service (QoS) and user preferences are used to rank the discovered services.

2. **Execution:**

a. **Process Mediation (Orchestration/Choreography)** resolves process heterogeneity between requester and provider. This may e.g. be needed when the provider service has a different granularity and multiple invocations need to be made to fulfill a goal. To achieve this, the choreography interfaces of both goal (requester) and service (provider) are processed, taking into account any concrete data. This data can be initially provided with the goal, changed by choreography rules, or received from a service. This subprocess can be seen as a conversation between requester and provider. There may be several steps involving updating the processing memory of the choreography component regarding requester or provider, mediating data, and invoking a service.

b. **Data Mediation** resolves data heterogeneity between requester and provider. It transforms instances of the ontologies known to the requester to instances of the ontologies known to the provider, or vice versa. Assuming there is a data heterogeneity problem, data mediation is performed two times: at the beginning, to convert the instances provided by the goal to instances of the web service; and at the end, to convert the response instances of the web service to instances comprehensible by the goal. (not used in swing)

c. **Service Invocation** and **Grounding**: The information which WSDL operation of a web service needs to be called with what instance data and the type of instances that is returned is specified in the grounding information of the WSMO web service description. For the actual invocation of the WSDL-based web service, the input data needs to be lowered from ontology instances to an XML-based message and later the response lifted to ontology instances.

It should be noted that none of the WSMO *xx*-Mediator conceptual elements are actually used in WSMX yet. They will be integrated at some later time. Nonetheless, process and data mediation are available; only the way they work might diverge in some details from the description in WSMO conceptual documents.

#### 4.2.6.3 WSML Language Elements

A semantic composition framework must provide meaningful service descriptions and enhance the composition process. In WSMO, the following elements are specific to descriptions of Geospatial Web Services:

*Non-functional properties* are used to describe service metadata similar to those specified by OGC Capability Documents.

*Pre-conditions* state that a valid request, for example an instance of for example a get-Feature request, must exist before the WFS service can be invoked.

*Post conditions* define constraints on the output of a service invocation. for example, if WFS is successfully executed, it will return a GML Feature Collection.

*Choreography* describes, in case of WFS, that each instance of a getFeature request causes an invocation of the OGC service and the result is translated into a WSML encoding of a GML Feature Collection. Adapters have to be defined, in order to implement Web Service requests from WSMX to OGC Services and to parse responses back into WSMX. Specific adapters can be built based on the generic Adapter Framework (Bussler et al., 2005).

*Orchestration* describes the interaction with further services.

Considering the WFS description as example, “getFeatureRequest” is modelled as a WSML concept, which is part of a globally available WFS ontology describing in- and outputs of WFSs. The concept “getFeatureRequest” is used in the state signature, which is part of the choreography of a WSML Web Service description; it provides the means for grounding a Web Service request to the actual service instance. The “FeatureCollection” concept is part of an extensive GML ontology that captures the GML semantics.

Application specific concepts are part of domain ontology. The example use case covers the domain of fire, meterology, plume dispersion and airports. Concepts of the domain ontology, like “Building”, "Airport" or “Runway”, are used to capture the semantics of application specific feature types, i.e. models for the geospatial data that is returned by the WFS. The link between the data model and the real world model is defined as an annotation (Klien et al., 2007; Schade et al., 2008; Grcar and Klien, 2007). A graphical tool guiding users in implementing such annotation has been developed (JSI) (Schade et al., 2008). More details about service and workflow semantics are described in section 5.7

**Defining Geospatial Task as WSMO Goal**

Capturing a user goal sufficiently is one of the main tasks in using frameworks such as WSMO. For instance, the application user might pose a question such as “Give me the recent weather conditions near airport X”. Such a goal must be formulated in a way that a service advertising such capability is found. The graphical tool that has been implemented for defining annotations can be reused for defining such goals (Andrei et al., 2008). The discovered service needs to be invoked based on available information and correct meterologic data for “airport X” have to be returned. In case no single service can satisfy the goal, services should be found that in combination resolve the user's goal.

A WSMO Goal for discovering and invoking of an anapplication logic resembles a service description, including both the capability and the interface section. The choreography of a Goal does not only express the users preferred mode of interaction with a WFS, it is also used in service discovery. Discovery is implemented using the WSMX reasoning engine IRIS (Vasiliu, 2006). Web Service descriptions are matched against the previously generated goals, which both follow the WFS annotation pattern as described in (Schade et al., 2008). A service will be selected if its capability and its choreography interface match that of the Goal. A case study of extending the discovery approach to processing services, especially WPS, can be found in (Hoffmann et al, 2008), and (Schade et al., 2008).

### 4.3 Asynchronous Web Services

The typical communication pattern of web services is *synchronous*, meaning a request is followed by a response. The HTTP protocol is a typical representative of this class. However, in some cases, especially long running computations, a requestor does not want to wait and be idle until the sever has finished the computation. In this case, the *asynchronous* communication pattern could be applied. The mainstream IT world has several protocols for this case such as:

- HTTPR
- JMS
- IBM MQSeries Messaging
- MS Messaging

In the OGC Web Service context, only the synchronous HTTP protocol is of interest. Two basic patterns are applicable here to allow asynchronous communication on top of HTTP and partly exercised in some specifications (e.g. WPS & WCS).

#### 4.3.1 Pull Model

The *pull* mechanism obliges the server to provide a reference back to the requestor on an initial request to allow the requestor to observe the status via this reference. Therefore, the requestor has to periodically call the server via this reference. This pattern has the advantage of requesting the status on demand but on the other side comes along with several disadvantages:

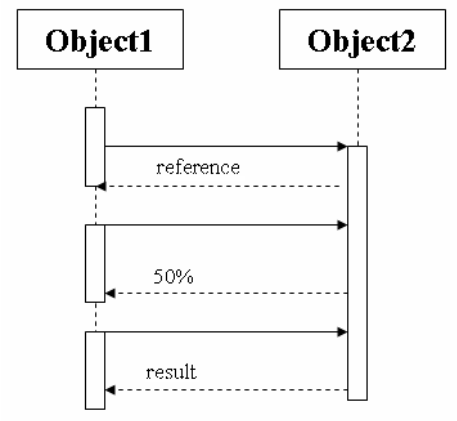



**Figure 11. Pull Model**

The message volume is increased, because for each status request, messages have to be exchanged. Besides, the requestor is not directly informed when the process has finished and might not notice when the process has finished because the requestor needs to request the status and will not get any status information automatically form the server.

### 4.3.2 Push Model

The *push* mechanism allows requestors to specify a callback endpoint which should be notified whenever the process has finished. This eliminates interim status requests to the server and thus reduces the communication traffic especially for long running processes as typically seen in Grid computing environments. Additionally, the *push* mechanism allows the integration of OGC services in mainstream IT asynchronous workflow environments and reuse by reusing existing standards and tools. The WS-Addressing (W3C 2003) specification is in this case the prime candidate.

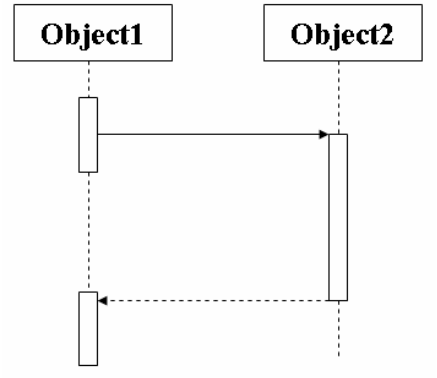



**Figure 12.** Push Model

As a disadvantage can be seen, that interim status messages are not directly foreseen by this pattern. A publish/subscribe mechanism would be one solution to this problem.

# 5 Developed Concepts

The following sections describe the concepts either considered or practically evaluated in the context of the OWS-6 Scenario.

## 5.1 Workflow Exposition

There are several ways to expose a workflow. For translucent and especially opaque workflows, many workflow engines are offered by the market. In order to exchange information between different workflow engines and use different workflow language via a single interface, interoperability and thereby standards come into the picture.

However, each vendor has its own interface or relies on standards from different standardization bodies. This section elaborates different ways of wrapping an opaque workflow in a standardized way and lays out the experience gained in the scenario exercised in this testbed.

### 5.1.1 Plain WS-*

One option to expose an opaque workflow is to apply mainstream IT standards. Particularly, standard web services invokable via HTTP GET, POST or SOAP have to be considered. To gain interoperability, common practice is to describe a web service with a Web Service Description Language (WSDL) document. This document explicitly describes the message structure for the input and output messages. The plain Web Service approach has the advantage of being compatible to mainstream IT enterprise architectures. However, plain Web Services are not based on OWS-Common and therefore do not fit in traditionally OGC based architectures. Additionally asynchronous invocation is not guaranteed and there is no standard about how the input and output message structure should look like which limits the level of interoperability.

### 5.1.2 Web Processing Service

The OGC Web Processing Service (WPS) Specification (OGC 2007) describes a standardized way to perform (geo) processes in SDIs. These processes can be as simple as the sum of two numbers (e.g. population) or as complex as a global climate model. The data required by the service can be delivered across a network or made available on a server. Image data formats or data exchange standards such as Geography Markup Language (GML) can be used for the resulting data.

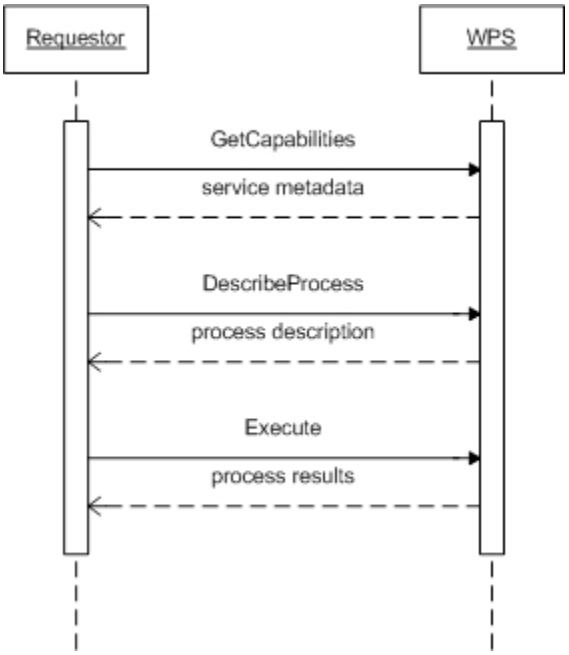



**Figure 13.** Basic WPS interaction

Therefore, the interface is held very generic and is mainly based on three operations as shown in figure 13. Since a WPS should be able to be integrated in to the OpenGIS framework, it has to offer the GetCapabilites operation generally described by the OWS Common specification. Besides the OWS Common metadata (see Section 2.3.4), and basic service metadata, all processes offered by the WPS are briefly described in the capabilities document. The GetCapabilities operation can only be invoked via HTTP-GET and the request should be Key-Value-Pair (KVP) encoded. A minimal request can be found in (OGC, 2007)

A requestor first requests the service metadata via the GetCapabilities operation. Detailed information is obtained by calling the DescribeProcess operation for an operation listed in the service metadata. Finally, the execute operation is invoked with all required input data identified by the DesribeProcess response. Especially, the flexibility of the WPS specification can be used for the workflow wrapping approach: All required workflow input data can be mapped to WPS input data.

Since any kind of calculation or algorithm can be exposed as a WPS process, the same principle can be applied for workflows.

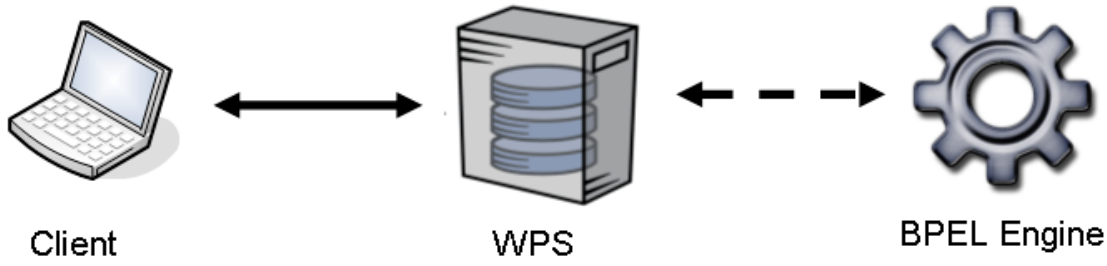



**Figure 14. Workflow Engine Wrapping Architecture**

As depicted in figure 14, the client sees the Workflow engine as a simple WPS process and is not necessarily aware of the underlying Workflow engine. The OGC Discussion Paper 08-123 discusses a transactional extension for the WPS, which applies a generic approach and a concrete profile for BPEL. This profile allows clients to dynamically deploy and undeploy workflows as WPS processes. In the case of OWS-6, a static-non transactional approach is applied which wires a WPS process directly to a workflow in a BPEL engine.

This architecture has the advantage, that different Workflow engines can be exposed as a WPS processes without changing the workflow appearance to the client. The client always can invoke a WPS process independent of the underlying workflow engine or workflow script language. Besides, by encapsulating a workflow engine with a WPS, the advantages of a WPS can be used, such as integration into the OGC service stack, asynchronous execution and basic message structure interoperability combined with the flexibility of input data description.

### 5.1.3 Web Coverage Service

A Web Coverage Service (WCS) allows standardized access to spatial raster data organized as coverage layers. In terms of wrapping a workflow as a WCS, only workflows that output raster data could be used. The workflow engine could be encapsulated by a WCS and exposed as a WCS coverage layer. Since the WCS specification only offers a well-structured and limited request schema, workflow input parameters not expressible as WCS parameters could not be used. Therefore, wrapping a workflow as WCS coverages has only a limited usage. However, since many clients can understand coverage data and can interact with a WCS, standardized access to (a limited class of workflows) is possible.

### 5.1.4 Web Feature Service

A Web Feature Service (WFS) allows standardized access to spatial vector data organized as features inside feature collections.

In principle, this approach faces similar limits to the WCS exposition described above.

In detail, once a composition is deployed to the execution engine, invocation is encapsulated in an OGC WFS. In cases where a composition does not require specific input parameters, i.e. all service requests are set within the composition, and where the execution results is a collection vector data, a WFS interface can be used for encapsulation. This approach has the advantage to allow any WFS client executing the complex composition. If a *getFeature* request is send to the encapsulating WFS, the service internally calls the execution and translates the result into a GML feature collection.

### 5.1.5 WF-XML

Wf-XML utilizes a loosely coupled, message-based approach to facilitate rapid implementation using existing technologies. It will describe the syntax of these messages in an open, standards-based fashion that allows for the definition of a structured, robust and customizable communications format.

Wf-XML can be used to implement three models of interoperability; specifically, chained workflows, nested workflows and parallel-synchronized workflows. Wf-XML supports these three types of interchanges both synchronously and asynchronously, and allows messages to be exchanged individually or in batch operations. Furthermore, this specification describes a language that is independent of any particular implementation mechanism, such as programming language, data transport mechanism, OS/hardware platform, etc.

However, Wf-XML has no SOAP binding defined and Wf-XML based services do not fit into the OGC service stack directly, because it e.g. does not support a GetCapabilities operation.

A RESTful Wf-XML (Wf-XML-R) developed in OWS-5 was used in this testbed, but but no specific implementation details were available to be included in this ER.

## 5.2 Data Transfer Patterns

Patterns for transferring and managing data have been studied extensively in the workflow research community. (see e.g. Russel et al., 2005) In terms of this testbed, only the transfer of data from the client to the workflow engine and from one workflow participant to another was relevant.

In this sense, two different aspects have to be regarded:

1. Keeping data in a centralized repository vs. keeping the data by the data generating entity

2. Sending data by value vs. send data by reference

Both aspects are discussed in the following sections.

### 5.2.1 Data Management

In the OWS-6 testbed, all data (input data to the workflow and to the services) was kept at the corresponding entities. In other words, one participant has sent the data which serves as input for another participant either by value or reference (see next sub section) to the workflow engine, which dispatches the data (reference) to the relevant workflow partner. No centralized repository was used where workflow partners store and access the data.

### 5.2.2 Data Transfer

#### 5.2.2.1 By Value

Sending data *by value* sends the whole data set from one participant to another. The workflow engines, as orchestrating entity, passes the data to the target participant. This concept has been proven useful only for small datasets and simple data structures, such as coordinate pairs, buffer distances etc, because the data has to pass the workflow engine and its pre determined communication channel. More efficient communication channels or timeframes could not be considered. Therefore, for larger datasets, sending data *by reference* seems to be more useful.

#### 5.2.2.2 By Reference

Sending data *by reference* does not send the actual dataset but only a pointer to the data. In the OWS-6 testbed, the data prevailed on the data generating services (e.g. WCS, WPS) and a URL was passed to fetch the data over the internet. The reference is passed via the workflow engine to the target participant. The actual data is then transferred via an external channel. This approach seems to be more applicable for large data sets, because, the data does not go through an intermediary and could be fetched via a chosen transport channel, which seems to be more efficient.

## 5.3 Workflow Response Delivery

As the last step in the lifecycle, the result data has to be delivered back to the initial requestor. Obviously, the response will be delivered according to

a) the interface of the workflow (see section 5.1)

b) the initial request based on a)

However, there are three different patterns for obtaining the result, which will be elaborated in the following sub sections.

### 5.3.1 Raw Data result

The data is delivered back as plain raw data. This pattern is related to Section 5.2.2.1 but is more strict in a sense that the data is delivered without any interface specific overhead. For instance, if a Geotiff image is requested, the image is delivered in binary format and not wrapped into other message structures.

For instance, WPS or WMS offer explicitly the option for delivering raw data but this concept can be used beyond these interfaces. In the OWS-6 testbed, this was especially useful for WPS delivering binary data, since the binary data had not to be encoded e.g. as base64 data. However, only one result can be delivered at a time.

### 5.3.2 Referenced Result

This pattern delivers only a reference to the data back in the meaning of section 5.2.2.2. In the OWS-6 testbed, this pattern was successfully combined with the previously described pattern: The raw data is referenced and only the reference is delivered back to the requestor. This has the advantage that multiple results could be returned back at once.

### 5.3.3 Encapsulated Result

Another option is to explicitly wrap the data in an interface specific encoding. For instance, gml features could be encapsulated in a wfs:FeatureCollection or in a ExecuteResponse document. This could be especially useful if the workflow is exposed via a known interface and only the specific interface response encoding is understood by the client.

In the OWS-6 testbed, the final workflow partner was a data providing service (WCS-T). The workflow engine stored all relevant results as new layers in that service and returned only a reference to the newly added layers. In other words, the results were encapsulated as WCS layers inside of the workflow and the requests and response message to fetch the data had to be according to this interface description.

## 5.4 Architectural Style Independent Workflows

**Development Environment**

The development environment is developed by SINTEF. It provides support for constructing (W3C and OGC) Web Services compositions and allows a user to annotate and discover Web Services including an extension for semantic information. The SINTEF Composition Studio has been chosen as the basis for the development environment because it is open source and operates on an abstract UML level when composing Web Services (Hoff et al., 2006).

**Top-level processes**

The main functionalities of the development environment are the creation of Web Service compositions, semantic annotation of Web Services and compositions, discovery of services, deployment of compositions and execution of compositions. Use of the development environment will typically only utilize parts of the available functionality. We will show two process models supported by the development environment.

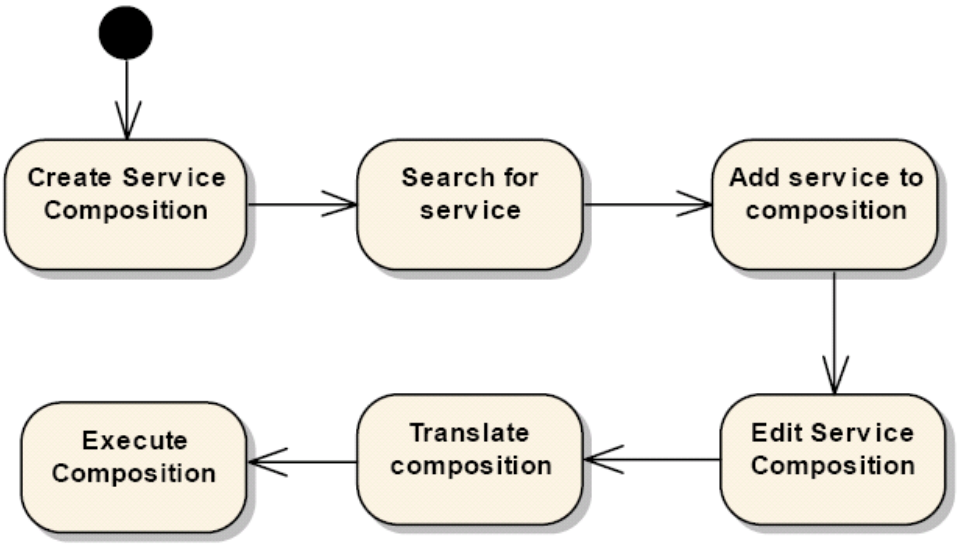



**Figure 15.** Composing and execution of a service composition.

Figure 14 shows the process of creating and testing a service composition. This process will typically start with the creation of a new, empty service composition. To populate the empty composition with existing services a search for services will be performed. The result of the search can then be added to the composition. The service composition will then be edited, for instance to describe the control flow. After finishing the composition it must be translated to a language the execution engine understands. To test the finished service composition it can be deployed on the execution engine and executed within the development environment. This will allow a service composition developer to have access to both composing and testing facilities in the same environment.

### Import of Web Feature and Processing Service

The Composition Studio has a local repository for resources transformed into UML representations, called Local Dictionary. With use of the import of service description menu it is possible to import a Web Feature Service. or WPS Figure 16 shows how to access the menu for importing WFS.

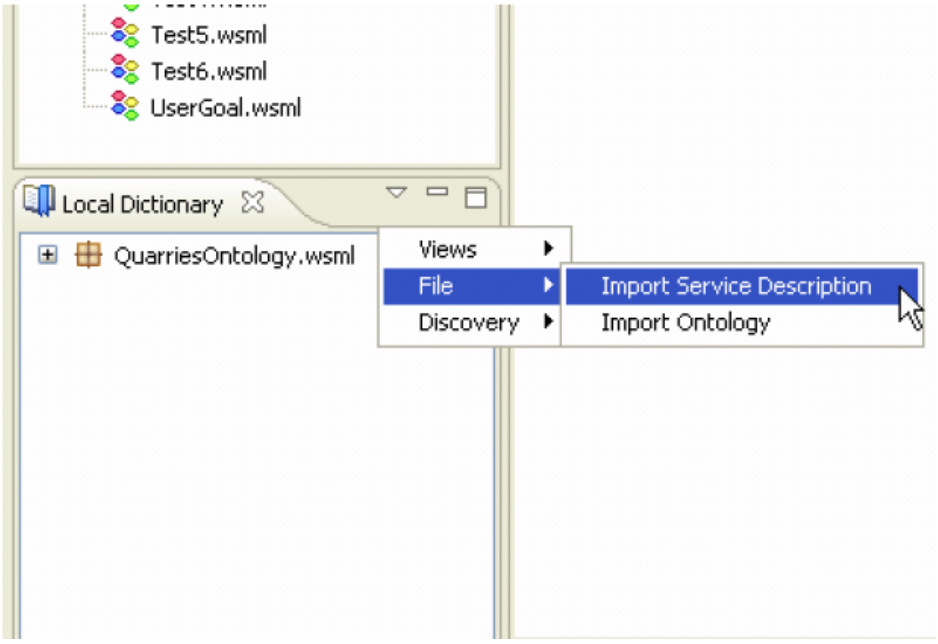



**Figure 16.** Import of Service Description.

After activating the import service description menu item a dialog window will appear, as seen in Figure 17. To import a WFS enter the base URL to the service. This will be used to parse the WFS description and create a UML representation in the Local Dictionary of the WFS. The local name is used for better naming of the UML resource.

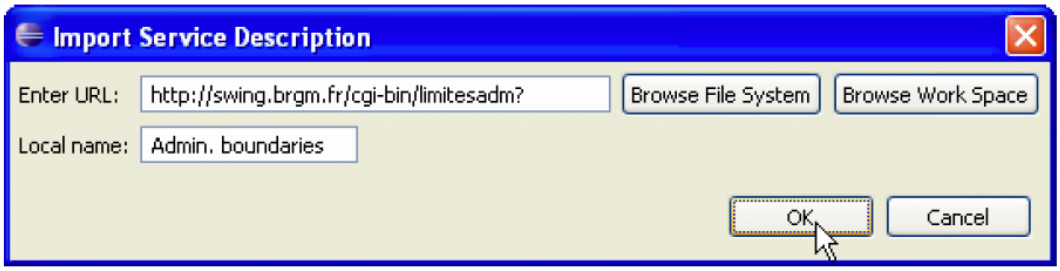



**Figure 17.** Enter an URL and a local name.

The transformation from WFS to UML is quite simple. Given the URL to the WFS the WFS document is parsed and transformed into a UML package containing a UML class with a WFS stereotype. Then each of the features of a WFS is transformed into a UML property contained in the UML class. Table 1 shows how a Web Feature Service is represented in UML.

| WFS | UML |
|---|---|
| Web Feature Service | Class with a WFS stereotype |
| Feature | Property with a Feature stereotype |

**Table 1.** An overview of the mapping between WFS and UML.

### Deployment of a composition

After a user has designed a composition, he or she must deploy the service so that it may be executed and discovered. Compositions may for example be stored , i.e. registered at a BPEL engine or (if including Semantic Web Technology) in WSMX. In order to store a composition in WSMX, the user must translate the composition into WSML, the Composition Studio has automatic support for translation. This creates a WSML file in the same project folder as the composition. The user may right-click on the WSML file (to open the context menu) and select menu option “SEE Store”. The development environment will open the window shown in Figure 18. This window allows the user to select the WSMX server that should receive the WSML-file. When user has selected the server, he may press the “OK” button to store the WSML file. He may also cancel the deployment of the WSML file by pressing the “Cancel” button. A similar export functionality is provided for BPEL.

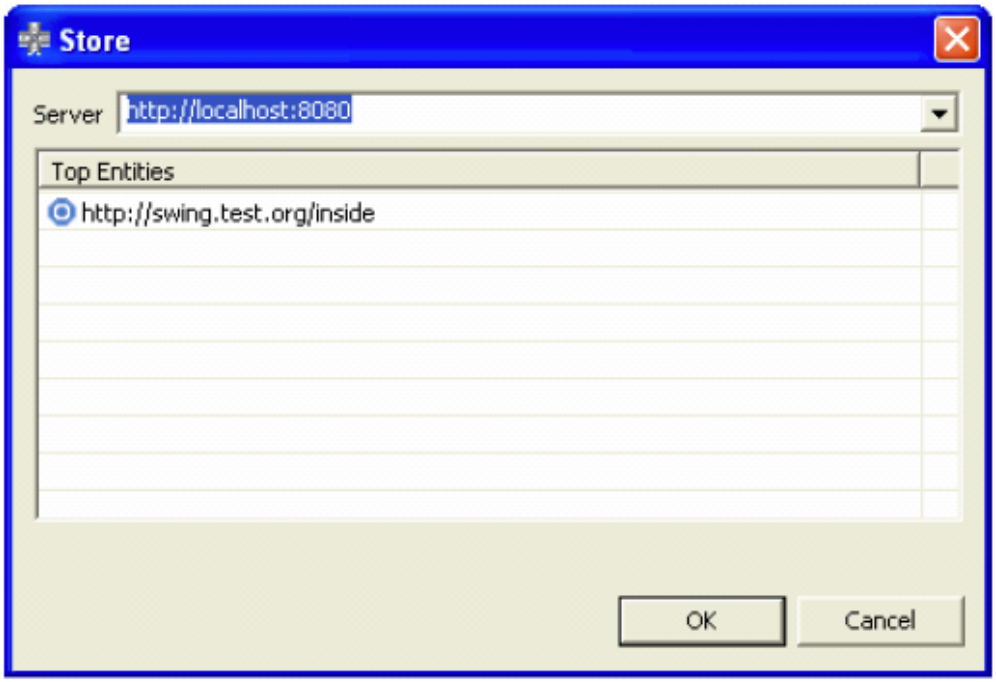



**Figure 18.** The deployment dialogue window.

### 5.4.1 BPMN vs. UML 2.0 Activity Diagrams

The reason for using UML Activity Diagrams in favour of BPMN diagrams is that UML does also provide profiles that makes it easy to extend models with specific fields and annotations. For complex project, often UML Class diagrams UML Sequence diagrams are necessary, which could be easily integrated with UML Activity Diagrams.

Additionally BPMN 1.0 does not support data flow as well as UML activity diagrams. However, in BPMN 2.0, there are plans to increase the BPMN support for data/information models, and we will then look further into how to use BPMN as an alternative to UML activity diagrams in the future.

## 5.5 Asynchronous Workflow Participants

Message exchange in workflows can be either synchronous or asynchronous. Synchronous workflows have been studied in previous testbeds (see Section 4).

This testbed focuses on asynchronous workflows, meaning in this scope, message exchange between the workflow engine and the workflow partner in an asynchronous fashion.

The workflow language and the workflow partners both have to support asynchronous message exchange. Since BPEL and WPS were the prominent representatives, this sections focus only on these technologies.

### 5.5.1 WPS-Push Model

Since the WPS was heavily used in the OWS-6 GPW exercise, the asynchronous capabilities of this service have to be analyzed. The WPS specification states in table 50 that the *status* attribute in the *ResponseDocument* element should be used to indicate an asynchronous pull request. The immediate status response would look for instance like:

```
1  <ns:ExecuteResponse xmlns:ns="http://www.opengis.net/wps/1.0.0" xmlns:xsi="
   http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="http://www.opengis.net/wps/1.0.0
   http://geoserver.itc.nl:8080/wps/schemas/wps/1.0.0/wpsExecute_response.xsd" serviceInstance="
   http://geoserver.itc.nl:8080/wps/WebProcessingService?SERVICE=GetCapabilities&amp;SERVICE=WPS"
   xml:lang="en-US" service="WPS" version="1.0.0" statusLocation="
   http://geoserver.itc.nl:8080/wps/RetrieveResultServlet?id=1239702719520">
2    <ns:Process ns:processVersion="2">
3      <ns1:Identifier xmlns:ns1="http://www.opengis.net/ows/1.1">
   org.n52.wps.server.algorithm.SimpleBufferAlgorithm</ns1:Identifier>
4      <ows:Title xmlns:wps="http://www.opengis.net/wps/1.0.0" xmlns:ows="http://www.opengis.net/ows/1.1"
   xmlns:xlink="http://www.w3.org/1999/xlink">Create a buffer around a polygon.</ows:Title>
5    </ns:Process>
6    <ns:Status creationTime="2009-04-14T11:51:59.520+02:00">
7      <ns:ProcessStarted percentCompleted="33"/>
8    </ns:Status>
9  </ns:ExecuteResponse>
```

**Listing 8. WPS Pull Response Message**

The statusLocation="http://geoserver.itc.nl:8080/wps/RetrieveResultServlet?id=1239702719520" attribute (line 1) can be used to *pull* additional status requests. Until process completion, the status message will not change except the optional percentCompleted attribute.

When the status URL is requested and the process has finished, the general WPS execute response as specified in (OGC 2007) will be delivered back. An example can be found in Listing 9.

```
1  <ns:ExecuteResponse xmlns:ns="http://www.opengis.net/wps/1.0.0" xmlns:xsi="
   http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="http://www.opengis.net/wps/1.0.0
   http://geoserver.itc.nl:8080/wps/schemas/wps/1.0.0/wpsExecute_response.xsd" serviceInstance="
   http://geoserver.itc.nl:8080/wps/WebProcessingService?SERVICE=GetCapabilities&amp;SERVICE=WPS" xml:lang="en-US"
    service="WPS" version="1.0.0" statusLocation="http://geoserver.itc.nl:8080/wps/RetrieveResultServlet?id=1239702719520">
2    <ns:Process ns:processVersion="2">
3      <ns1:Identifier xmlns:ns1="http://www.opengis.net/ows/1.1">org.n52.wps.server.algorithm.SimpleBufferAlgorithm</
   ns1:Identifier>
4      <ows:Title xmlns:wps="http://www.opengis.net/wps/1.0.0" xmlns:ows="http://www.opengis.net/ows/1.1" xmlns:xlink="
   http://www.w3.org/1999/xlink">Create a buffer around a polygon.</ows:Title>
5    </ns:Process>
6    <ns:Status creationTime="2009-04-14T11:51:59.520+02:00">
7      <ns:ProcessSucceeded>Process successful</ns:ProcessSucceeded>
8    </ns:Status>
9    <ns:ProcessOutputs>
10     <ns:Output>
11       <ns1:Identifier xmlns:ns1="http://www.opengis.net/ows/1.1">result</ns1:Identifier>
12       <ows:Title xmlns:wps="http://www.opengis.net/wps/1.0.0" xmlns:ows="http://www.opengis.net/ows/1.1" xmlns:xlink
   ="http://www.w3.org/1999/xlink">Buffered Polygon</ows:Title>
13       <ns:Data>
14         <ns:ComplexData schema="http://geoserver.itc.nl:8080/wps/schemas/gml/2.1.2/gmlpacket.xsd">
15           <pac:GMLPacket xmlns:pac="http://www.opengis.net/examples/packet">
16             <pac:packetMember>
17               <pac:StaticFeature>
18                 <gml:polygonProperty xmlns:gml="http://www.opengis.net/gml">
19                   <gml:Polygon>
20                     <gml:outerBoundaryIs>
21                       <gml:LinearRing>
22                         <gml:coord>
23                           <gml:X>129.1855204324875</gml:X>
24                           <gml:Y>-50.109081932755714</gml:Y>
25                         </gml:coord>
26                         [...]
```

**Listing 9. WPS Pull Final response.**

In order to invoke such a pull based WPS with BPEL, the principle above could be applied by frequently checking the statusLocation in a loop (lines 12-17, Listing 10) over an invoke (line 16, Listing 10) statement, as shown in Listing 10.

```
1   <! – invoke WPS_Plume service  -->
2   <invoke name="Invoke_Plume" partnerLink="wps_plume" portType="ns1:ExecutePortType" operation="Execute"
    inputVariable="Invoke_Plume_Execute_InputVariable1" outputVariable="Invoke_Plume_Execute_OutputVariable1"/>
3   <! --  Because there is no operation for checking service status in WPS, a service (WPSStatus) which is used to check
    service status is added here. Its input is status location, and output is WPS response document-->
4   <! – assign the status location of plume service to the input of WPSStatus service  -->
5   <assign name="Assign_Plume_Status_Location">
6     <copy>
7       <from variable="Invoke_Plume_Execute_OutputVariable1" part="OutPart" query="
    /ns1:ExecuteResponse/@statusLocation"/>
8       <to variable="Invoke_WPSStatus_CheckStatus_InputVariable" part="payload" query="
    /ns7:CheckStatusRequest/ns7:statusLocation"/>
9     </copy>
10  </assign>
11  <! – loop until process succeeded  -->
12  <while name="While_Plume" condition="ora:countNodes('Invoke_WPSStatus_CheckStatus_OutputVariable', 'payload',
    '/ns1:ExecuteResponse/ns1:status/ns1: ProcessSucceeded' = 0  ">
13  <! – wait for 2 minutes  -->
14  <wait name="Wait_Plume" for="'PT2M'"/>
15  <! – invoke WPSStatus service to get service status  -->
16  <invoke name="Invoke_WPSStatus" partnerLink="wps_status" portType="ns7:CheckStatusPortType" operation="
    checkStatus" inputVariable=" Invoke_WPSStatus_CheckStatus_InputVariable " outputVariable="
    Invoke_WPSStatus_CheckStatus_OutputVariable"/>
17  </while>
18  <! --  assign plume output to WCS-T-->
19
20  <assign name="Assign_WCST">
21  <copy>
22      <from variable="Invoke_Plume_Execute_OutputVariable1" part="payload" query="
    /ns1:ExecuteResponse/ns1:ProcessOutputs/ns1:Output[1]/ns1:Reference/@href"/>
23      <to variable="Invoke_WCST_transactionOperation_InputVariable" part="inParameters" query="
    /ns7:Transaction/ns7:InputCoverages/ns7:Coverage/ns7:Reference/@ns5:href"/>
24    </copy>
```

**Listing 10.** BPEL mechanism for WPS pull invocation.

Unfortunately, push support (see Section 4.3.1) is not yet specified in the current WPS specification.

There are generally two approaches to solve this problem:

#### 5.5.1.1 Extending the WPS execute request

One option would be to directly extend the WPS request. The *ResponseDocument* would get two optional attributes in the <wps:ResponseDocument> element

- callbackAddress

this attribute points to an endpoint where the process result is pushed to upon completion.

- callbackType

this attribute specifies the callback mechanism type. An *urn* shall be used which is indicated in the extended Capabilities metadata. To indicate the supported callback mechanisms, the service metadata should be extended with the following structure:

```
<wps:Capabilities>
    [...]
    <wps:CallbackOfferings>
        <wps:CallbackType>urn:ogc:wps:async:push:url</wps:CallbackType>
        <wps:CallbackType>urn:ogc:wps:async:push:email</wps:CallbackType>
        <wps:CallbackType>urn:ogc:wps:async:push:skype</wps:CallbackType>
        [...]
    </wps:CallbackOfferings>
</wps:Capabilities>
```

**Listing 11. WPS Capabilities Extension**

A <wps:CallbackOfferings> element has 0..n <wps:CallbackType> each containing a urn.

For instance, the urn:ogc:wps:async:push:url will be used to indicate that the process results shall be send to the url stated in the callbackAddress attribute.

Both attribute have always to be used in conjunction. The *status* attribute has to be therefore always turned to *false*.

#### 5.5.1.2 Use optional message parts

WS-Addressing (OASIS, 2004) is a mainstream IT solution to this problem. The OASIS specification utilized the SOAP header to send a XML structure containing a callback address. This callback address should be used to push the data back, when the process has finished.

A typical example can be seen in listing 12.

```
<soapenv:Header>
  [...]
    <wsa:To>http://localhost:8080/52n-wps-webapp-2.0-rc2-SNAPSHOT/services/WPS</wsa:To>
    <wsa:ReplyTo>
      <wsa:Address>http://localhost:8080/wsaServlet/wsaServlet</wsa:Address>
    </wsa:ReplyTo>
    <wsa:MessageID>urn:uuid:F3F6FFDBC2641A1CEF1238579166033</wsa:MessageID>
    <wsa:Action>urn:execute</wsa:Action>
  </soapenv:Header>
  [...]
```

**Listing 12. WS-Addressing Header Example**

The WS-Addressing specification specifies its elements in detail. In the context of the WPS specification, if a WS-Addressing header is used, the expected response message to a request will be delivered to a the given address stated in the <wsa:Address> element. Therefore, this mechanism can be used in conjunction with the request methods outlined in the WPS specification.

In BPEL, the pattern described in listing 13 could be used to invoke a service with WS-Adressing.

```
1  <sequence name="main">
2      <!-- Receive input from requestor. (Note: This maps to operation defined in ows6gpwasync.wsdl) -->
3      <receive name="receiveInput" partnerLink="client"
4          portType="client:ows6gpwasync" operation="initiate"
5          variable=" Invoke_Plume_Execute_InputVariable " createInstance="yes"/>
6      <!--
7          Asynchronous callback to the requester. (Note: the callback location and correlation id is transparently handled
   using WS-addressing.)
8      -->
9      <invoke name="Invoke_Plume" partnerLink="WPS_Plume"
10         portType="ns1:ExecutePortType" operation="Execute"
11         inputVariable="Invoke_Plume_Execute_InputVariable"
12         outputVariable="Invoke_Plume_Execute_OutputVariable"/>
13     <invoke name="callbackClient" partnerLink="client"
14         portType="client:ows6gpwasyncCallback" operation="onResult"
15         inputVariable=" Invoke_Plume_Execute_OutputVariable "/>
16   </sequence>
```

**Listing 13. WS-Addressing with BPEL**

#### 5.5.1.3 Comparison of both approaches

Both presented approach have certain advantages and disadvantages. Table 2 gives an overview:

| Criteria | WPS Inline Approach | WS-Addressing |
|---|---|---|
| Use within mainstream IT-architectures | - | + |
| SOAP | + | + |
| HTTP-POST | + | - |
| HTTP-GET | (+) | - |

| | | |
|---|---|---|
| Publish/Subscribe | - | + |

**Table 2.** Comparison of WS-Addressing and OGC approach

Enterprise IT architectures beyond the OGC scope uses in most cases the WS-* stack. WS-Addressing is used in this context for asynchronous message exchange. Therefore, utilizing WS-Addressing would foster the integration of OGC services in Enterprise IT architectures. On the other side, an OGC approach of extending the WPS schemata would not be beneficial for this goal.

Both approaches could be used with a SOAP binding, but when it comes to HTTP-POST and HTTP-GET the WS-Addressing approach fails, due to its orientation on the SOAP header.

Newer developments in the direction of event architecture publish/subscribe mechanisms for automated status updates require additional technology such as WS-Notification. For further details see the OWS-6 Event Architecture Engineering Report (OGC 2009a).

Another topic is the use of different message protocols than HTTP. By utilizing the WPS inline approach, different protocols could be requested such as Skype or email (SMTP) depended on the service offerings. Nevertheless, the WPS inline approach offers a greater flexibility in this direction.

### 5.6 Security Aspects in Workflows

Workflows are a powerful mechanism to efficiently streamline business processes. The delegation of certain processing steps of a business process to external partners fosters the quick adaptation of changing environments and leads to higher flexibility and scalability of those processes. Research in this field identified standardization as the key factor to take advantage of economies of scale and economies of scope. Standardized OGC Web Services enable the required interoperability among software of different vendors to allow the creation of workflows across enterprise boundaries.

However, partners will only conduct business if their (geo)rights, trust and security requirements are met.

In the context of OGC testbeds, experimental workflows have been developed and first attempts were made to address security aspects. Nevertheless, no generic solution could be found for security in workflows (e.g. the OWS-5 Testbed only regarded a RESTful security workflow approach which is not compatible to the OGC Security Architecture developed in the same testbed nor to Business to Business SOA based workflows also developed in OWS-5, see section 4.1).

The OWS-6 Security Engineering report (OGC 2009b) describes in detail approaches for securing plain OGC Web Services. Workflows that incorporate those secured OWS from different domains/enterprises have to deal with additional problems. This testbed focused especially on the aspect of trust between different domains (see OGC 2009b) and the delegation of authority between different domains.

Figure 19 visualizes this problem conceptually.

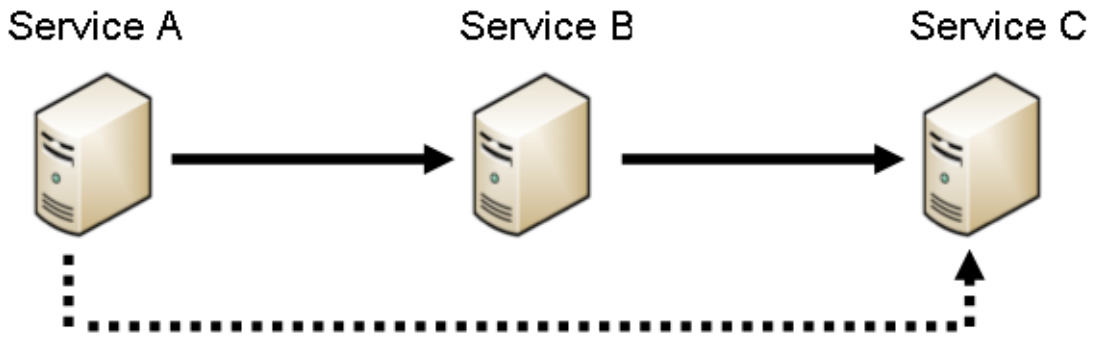



**Figure 19.** Conceptual Delegation Problem

Service A is known and trusted by Service C and furthermore has rights to access Service C. Service B is neither known nor trusted or has any rights to access Service C. In order to allow Service B to act on Service A's behalf to access Service C restricted to only dedicated resources more than simple trust is required: The delegation of authority. In this context, Service A has the *delegator* role, Service B is the *delegate* and Service C is the *targeted Service*.

In general, the concept of delegation has been studied e.g. in (Gasser and McDermott 1990) and with regard to role based access control, e.g. in (Zhang et al, 2003) In principle, a *delegator* delegates authority to a *delegatee*, which can act on behalf of the delegator. The delegatee's rights could be constrained for a dedicated resource. Therefore, delegation consist of the tuple

*(Delegator, Delegatee, Resources, Constraints)*

Additionally, two types of delegation could be distinguished: Direct delegation and indirect delegation.

In direct legation, the delegator *A* directly delegates the authority to a delegatee *B* to act on his behalf in regards to resource *C*. Resulting in

(A, B,C,{})

Indirect delegation lets B delegate the gained authority from A further to $B_1$. In this context, B becomes the delegator and $B_1$ the delegatee. This leads to

(A,(B,$B_1$,C,{}),C,{})

and could be regarded as delegation chains (figure 4).

An applicable concept for geoprocessing workflows with special regards to OGC web service is still missing. For the o the Grid computing world, proxy certificates were introduced (Welch et al., 2004), but have not been accepted and supported widely.

Besides, Wang (2005) proposed to extend SAML and use attribute statements, because SAML is widely accepted. However, the presented approach is not applicable to geoprocessing workflows, because the special requirements of OGC web services and workflows are not regarded. This results mainly in leaving open, how service should announce the requirements of such tokens, the non-spatial related constrainability and use of timestamps as security mechanism which are not useful in workflows, since the execution time is often not predicable.

The work presented in this paper picks up the basic idea of using SAML, but takes a different and more flexible path applicable to the OGC world and geoprocessing workflows as explained in the following section.

For enterprise architectures, WS-Trust and WS-Federation focus only on trust issues and not on full delegation of authority. Employing WS-Secure Conversation could be one solution to establish sessions (via key negotiation) but can be problematic in terms of delegation chains and non-repudiation, since only 1:1 relationships are established.

A solution for workflows composed of loosely coupled OGC Web Services is still missing.

In the context of the three different workflow architectures introduced in section 4.2

### 5.6.1 Transparent

In the transparent case, the client directly orchestrates the services. Hence, only direct interactions between the client and the workflow partner services exist and no delegation is required. The client needs to be trusted and needs to have access rights on each workflow partner service. Security aspects for direct interactions are covered by the OWS-6 Security Engineering Report.

### 5.6.2 Translucent

In the translucent case, a workflow engine is responsible for orchestration and thus invoking the services. Even though, the client knows which services are involved, the general problem here is that a Workflow Engine, which might have no rights to access a service on its own, has to invoke that service. Mapped to figure 20, the client is Service A, the workflow engine Service B and the secured OWS is service C.

In principle the attempt to invoke a secured service, where the secured service is not able to indentify the requestor because there is either no trust relationship or the requestor has no rights to access the secured service, would fail.

Therefore, the workflow engine (Service B) has to act on behalf of the initial requestor (Service A) which needs a trust relationship to the targeted secured service (Service C) and access rights as visualized in figure 18.

To solve this problem, figure 20 shows a generic approach to this problem.

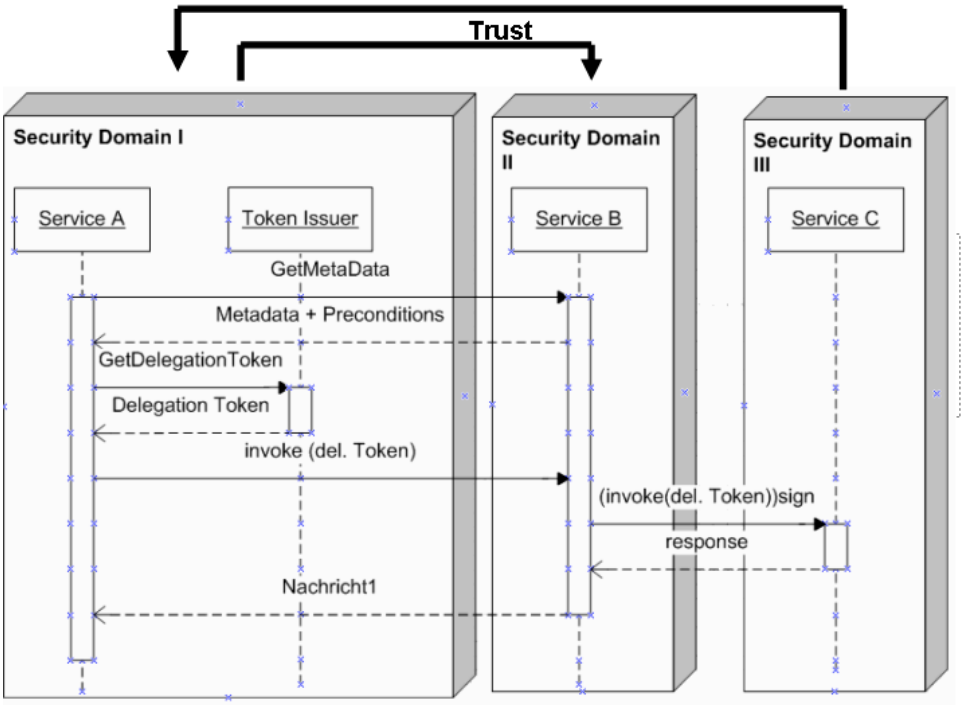



**Figure 20.** Generic Delegation Concept for OWS

#### 5.6.2.1 Preconditions

In a workflow scenario, the first step is to ask the WFE for service metadata and preconditions. For OGC Web Service, the GetCapabilties operation common to all OGC Web Services would be the first step. An alternative or additional way would be to use the WS-Metadataexchange (BAE Systems, IBM, Microsoft, IBM 2004) method to obtain service metadata like it is exercised in the mainstream IT-world. In one way or the other, metadata have to be collected from the WFE.

Preconditions play a key role in security enabled web services and therefore in GeoRM enabled OWS as well. In general, preconditions publicly announce a potential Web Service requestor, which conditions (in the context of security-which security model,

tokens, encryption mechanism) are required/supported. This concept ensures interoperability by allowing services to fulfil all required preconditions prior to the service invocation. A general workflow should follow the principle depicted in 20 according to (Kanneganti and Ramarao 2008).

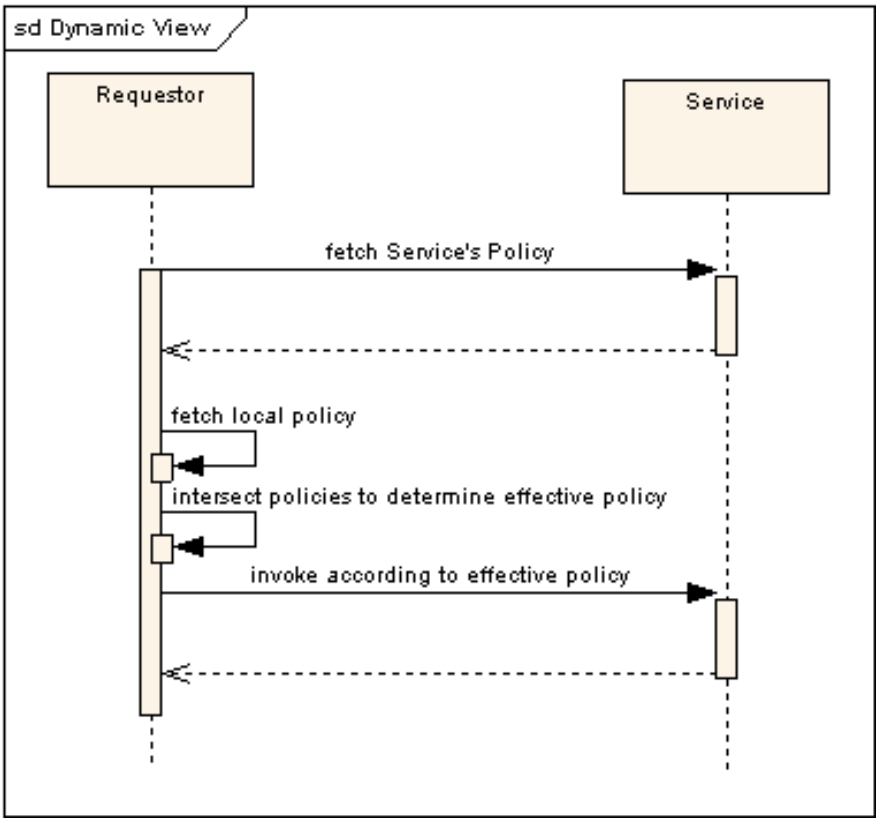



**Figure 21.** Precondition Processing

In terms of the using a GetCapabilities approach, a solution for indicating preconditions is to extend the GetCapabilities response with the following structure under the root element:

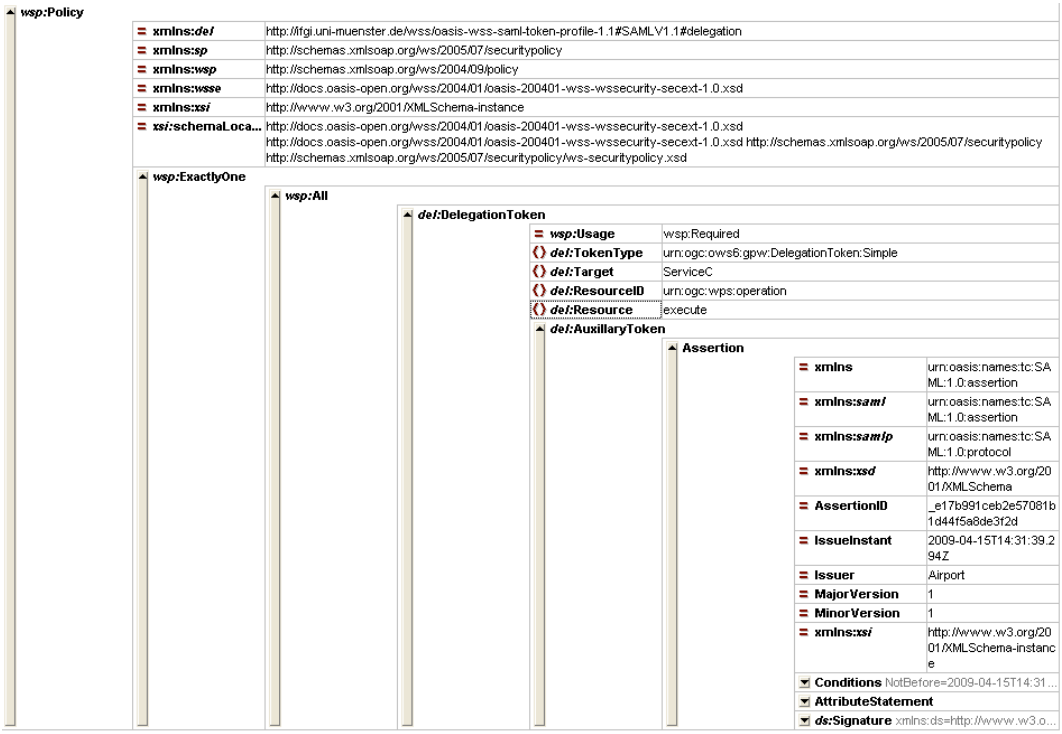
```
<PreconditionList>
  <Precondition type="urn:oasis:WS-Policy">
    http://foo/bar/ws-policy-precondiciom.xml
  </Precondition>
</PreconditionList>
```

**Listing 14.** Preconditions Extension

A <PreconditionList> element which serves as a container for 1..n <Precondition> elements. Each of these elements has a mandatory *type* attribute, which should contain a urn indicating the encoding of the document represented by the url given as the node value. In listing 14, *urn:oasis:WS-Policy* indicates a WS-Policy encoding for the document represented by *http://foo/bar/ws-policy-precondin.xml* url.

Other encodings are also possible. However, the preconditions have to state that the workflow engine requires a delegation token in order to access the secured (and known to the client) service.

A WS-Policy precondition encoding for a delegation token could follow the pattern described in listing 15.




**Listing 15.** WS-Policy Precondition

The WS-Policy document follows the OASIS WS-Policy and WS-SecurityPolicy specifications. To enable delegation, a DelegationToken element was stated in the first and only policy. This extension of WS-Policy is anticipated by WS-Policy and WS-SecurityPolicy to allow a flexible use of the specification.

In this case, the policy is marked as *Required* by using the *wsp:Usage* Attribute. The token type is given by the *del:TokenType* element. In the example from figure 22, a delegation token is indicated by the urn:

*urn:ogc:ows6:gpw:DelegationToken:Simple*

which in this case requires a simple delegation token (SDT).

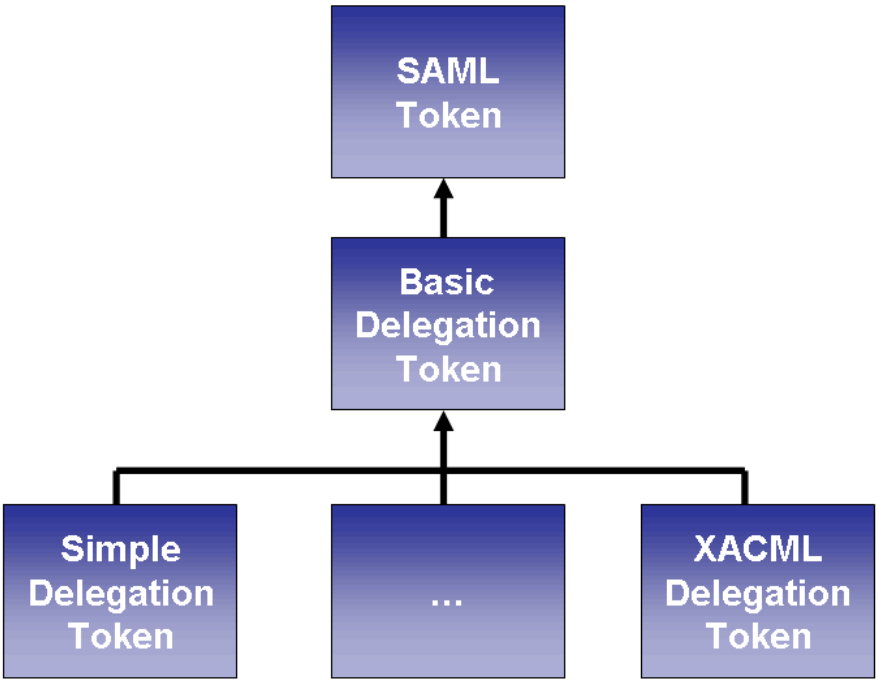



**Figure 22.** Delegation Token Hierarchy

In this testbed, a basic delegation token was engineered which servers as an anchor point for multiple extensions for specific requirements (see section 5.6.2.2)

In order to identify the target service, which should be invoked on behalf of the preconditions issuing service, the *del:Target* element is used. This element should contain the URL of the targeted service.

The del:ResourceID element should be used to indicate the type of resource requested by the delegate. Again, a urn is used. In the example used in this testbed, a WPS operation is indicated. This could be extended to different OWS types and resources.

The del:Resource element could be used to specify the actual resource. Since a wps operation is indicated by the urn given in the ResouceID element, the actual operation is stated here. This concept is also held flexible to allow different types and georesources, such as layers for e.g. WMS etc.

As the last element, an del:AuxillaryToken is provides a SAML assertion which serves as an identity token of the delegatee. It also includes a public key of the delegator, which

can be used by the target services to ensure non repudiation if this identity token is signed by a trusted authority such as the delegator.

#### 5.6.2.2 Delegation Token Issuing

After the client has received the service metadata and has fetched the preconditions indicating a required delegation token, it can proceed to obtain that token. In this testbed, a SecuredTokenServices (STS) defined by WS-Trust was used to issue identity tokens. Because of the WS-Trust specification is held general, it could be extended to issue different tokens such as delegation tokens.

Therefore, an extension was developed for issuing delegation tokens. As depicted in figure 22, all delegation tokens are based on the well known and established SAML (1.1) format to allow COTS software to read and extract the relevant information easily. The SAML basis was extended by a *basic delegation token (*BDT). A BDT includes all relevant information except access rights restrictions. The restriction could be handled by different BDT profiles such as for instance a potential (Geo)XACML Delegation Token Profile. In this testbed only a Simple Delegation Token (SDT) as a profile was implemented but as indicated before the approach is held flexible to allow other profiles such as XACML.

**Simple Delegation Token (SDT) Request**

Listing 16 shows the essentials of Simple Delegation Token sample request.

```
1   <wst:RequestSecurityToken xmlns:wst="http://schemas.xmlsoap.org/ws/2005/02/trust">
2     <wst:RequestType>http://schemas.xmlsoap.org/ws/2005/02/trust/Issue</wst:RequestType>
3     <wst:Lifetime>
7     <wst:TokenType>urn:ows6:gpw:DelegationToken:Simple</wst:TokenType>
8     <wst:KeyType>http://schemas.xmlsoap.org/ws/2005/02/trust/SymmetricKey</wst:KeyType>
9     <wst:KeySize>256</wst:KeySize>
10    <wst:Delegatable>false</wst:Delegatable>
11    <del:Delegatator xmlns:del="http://ifgi.uni-muenster.de/wss/oasis-wss-saml-token-profile-1.1#SAMLV1.1#delegation">
77    <del:Delegatee xmlns:del="http://ifgi.uni-muenster.de/wss/oasis-wss-saml-token-profile-1.1#SAMLV1.1#delegation">
144   <del:Transaction xmlns:del="http://ifgi.uni-muenster.de/wss/oasis-wss-saml-token-profile-1.1#SAMLV1.1#delegation">
145       <del:TransactionID>123</del:TransactionID>
146       <del:Target>foo.bar</del:Target>
147       <del:Resource>urn:foo:bar</del:Resource>
148       <del:ResourceID>123foo</del:ResourceID>
149   </del:Resource>
150   <del:Rights xmlns:del="http://ifgi.uni-muenster.de/wss/oasis-wss-saml-token-profile-1.1#SAMLV1.1#delegation">
151   urn:ogc:def:delegation:rights:absolute:one-time-use
152   </del:Rights>
153   </wst:RequestSecurityToken>
```

**Listing 16.** Simple Delegation Token sample request

The wst:RequestSecurityToken (line 1) defined by WS-Trust serves as the container for the SDT request. Line 7 requests the token type

*urn:ows6:gpw:DelegationToken:Simple*

which is requested by the delegatee and stated in the preconditions.

The extended elements are:

*Delegateable (line 10)*: Boolean value indicating whether or not the delegatee is allowed to delegate the token further. This could lead to delegation chains as depicted in figure 23.

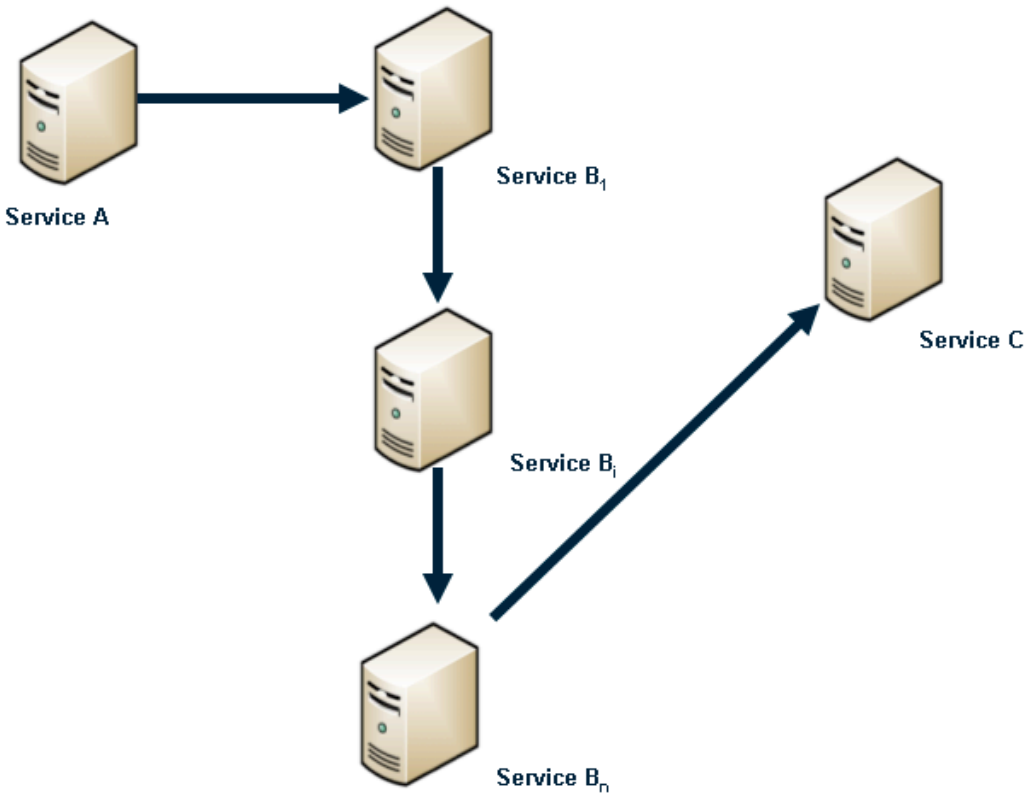



Figure 23. Delegation Chain

Delegator Service A could delegate the authority for a specific resource on Service C to Service $B_i$ which can delegate it further Service $B_{i+1}$ etc. In case this element is set to *true*, further delegators have to put the received delegation token as the delegator node value analogous to section II. B. For instance, if Service $B_i$ delegates it authority (which was delegated to Service $B_i$ by Service $B_{i-1}$) to Service $B_{i+1}$, the SDT issued by $B_{i-1}$ should go in there to allow Service C to determine that Service $B_{i+1}$ is acting on behalf of $B_i$ acting on behalf of $B_{i-1}$ .

*Delegator (line 11)*: Contains an identity token for the delegator, in this case a SAML Assertion, which could be fetched previously from a STS. This element is optional, if not provided, the delegation token issuing STS should embed the identity information directly based on other authentication means of the requestor. Other identity tokens could also be possible, but SAML was chosen, because it is widely accepted and supported.

*Delegatee (line 77)*: Contains an identity token for the delegatee, in this case a SAML Assertion, which was extracted from the preconditions.

*Transaction (line 144)*: The transaction has four parts:

*TransactionID*: UUID indicating the whole transaction with the delegatee. The main purpose is to prevent replay attacks. If a TransactionID has been previously used it should not be accepted by the target anymore as long as not stated differently in the *rights* section.

*Target*: Contains the target URL which was extracted from the preconditions.

*Resource*: Contains the actual resource which was extracted from the preconditions.

*ResourceID*: Contains ResourceID urn which was extracted from the preconditions.

*Rights (line 150)*: The simple delegation token adds the *Rights* section to the basic delegation token. In case of the simple delegation token, URNs are defined indicating certain rights. Up to now only:

*urn:ogc:def:delegation:rights:absolute:one-time-use*

is defined which grants full access but the TransactionIDs have to be accepted only once. This concept is also held flexible in order to allow different rights encodings, such as (Geo)XACML.

**Simple Delegation Token (SDT) Response**

The STS responds with a SDT. A sample token is shown in listing 17. The STS basically wraps all input data in a single token and signs the token. The whole token is delivered back.

The delegatee is transformed to the saml:subject element native for SAML assertion. Thereby, the SDT SAML assertion is only valid for the delegatee.

When it comes to revocation of delegated authorities (which is not covered in this ER), issuing only references would be the better solution because entities that want to validate the token have to request it which will only be possible if the token is still valid.

```
1   <Assertion [...]>
2     <Conditions NotBefore="2009-04-16T14:05:38.473Z" NotOnOrAfter="2009-06-16T14:10:38.473Z"/>
3     <AttributeStatement>
4       <Subject>
22      <Attribute AttributeName="Delegatable" AttributeNamespace="http://ifgi.uni-muenster.de/wss/oasis-wss-saml-token-profile-1.1#SAMLV1.1#delegatic
25      <Attribute AttributeName="Delegatator" AttributeNamespace="http://ifgi.uni-muenster.de/wss/oasis-wss-saml-token-profile-1.1#SAMLV1.1#delegatio
95      <Attribute AttributeName="Rights" AttributeNamespace="http://ifgi.uni-muenster.de/wss/oasis-wss-saml-token-profile-1.1#SAMLV1.1#delegation">
98      <Attribute AttributeName="Transaction" AttributeNamespace="http://ifgi.uni-muenster.de/wss/oasis-wss-saml-token-profile-1.1#SAMLV1.1#delegatic
108     <</AttributeStatement>
109   <ds:Signature xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
147 </Assertion>
```

**Listing 17.** Simple Delegation Token example

Please note that the SDT can include a del:DelegationChain element, which would contain previously issued SDT tokens in that chain. For instance, if Service $B_i$ delegates it authority (which was delegated to Service $B_i$ by Service $B_{i-1}$) to Service $B_{i+1}$, the SDT issued by $B_{i-1}$ should go in there to allow Service C to determine that Service $B_{i+1}$ is acting on behalf of $B_i$ acting on behalf of $B_{i-1}$ .

Furthermore, the Target, Resource and ResourceID elements are optional since they could be encoded directly in the rights section.

#### 5.6.2.3 Authentication with a Delegation Token

Once the secured target Service C is invoked by Service $B_n$ (figure 23) on behalf of Service A, with a signed request including the SDT, Service C has to authenticate the requesting service. Since Service $B_n$ is unknown and untrusted to Service C, Service C cannot authenticate Service $B_n$. In a role based access control system the request would be rejected.

However, a SDT tells Service C, that it does not need to know or trust service $B_n$. Only the predecessor (or respectively the known root of the delegation chain). Therefore, Service C can authenticate Service $B_n$ by means of Service A. Therefore, the SDT is extracted, and the subject is analyzed. Since the SDT is issued by a trusted party, the public key included in the SAML *subject* element belonging to Service $B_n$ could be extracted and trusted transitively. The public key could be used to validate the signature of the incoming request. Thus, $B_n$ could be authenticated and also Service A, which identity token is included in the SAML *del:delegator* attribute. For delegation chains, this process has to be repeated recursively.

#### 5.6.2.4 Authorization with a Delegation Token

After the requesting entity is authenticated, authorization is required. In RBAC systems, this is based on roles assigned to identities. With the delegation approach, it becomes possible to restrict access by the rights element in the SDT and delegate authority to a priori unknown entities. Therefore, Service C has to extract, understand and enforce the rights stated in the SDT token.

Interoperability is ensured, since Service C would announce which token types and profiles it can understand in its preconditions. These preconditions would be collected statically or dynamically by Service B and exposed to Service A which has to gather the required token in order to request the Services.

### 5.6.3 Opaque

Opaque workflows do not expose the workflow participants to the client. Therefore, the workflow orchestrating entity (workflow engine) is required to have access rights on each workflow participant analogous to the transparent case. However, it is thinkable that a workflow engine does not have access rights on each workflow participant, like in the OWS-6 Airport scenario. In this case, the workflow engine has to require a delegation token for this particular service according to the translucent case. Thus, the workflow is not anymore opaque and not totally translucent. *Hybrid* or *selectively translucent* would be the right terms to describe this kind of workflow.

## 5.7 Workflow Semantics

### 5.7.1 Requirements for WEBSERVICE specification for Geographic Information Services

*Semantic Annotation* refers to making the semantics of the service's underlying functionality or data explicit by establishing a link to Domain Ontologies. The goal of the annotation process is to generate a WSMO WEBSERVICE (written in WSML) for a specific OGC service that integrates explicit semantic descriptions of the functionality or data that is served. Figure 23 gives an overview on all the items that are involved in the process of annotating WFS in WSMX. Everything starts and depends on the lower right of the picture: the real-world entities, which are represented as spatial information objects. These spatial information objects are encoded as features in the Geographic Markup Language (GML) and served via OGC data services. In the following we will concentrate on the requirements for annotating WFS, but WCS and WMS can be described according to the same principles. In brief, first service documents are translated into WSML, second, links to Domain Ontologies are established.

The effects of semantic annotation on OGC standards are currently discussed within OGC (Duchesne et al, 2008).

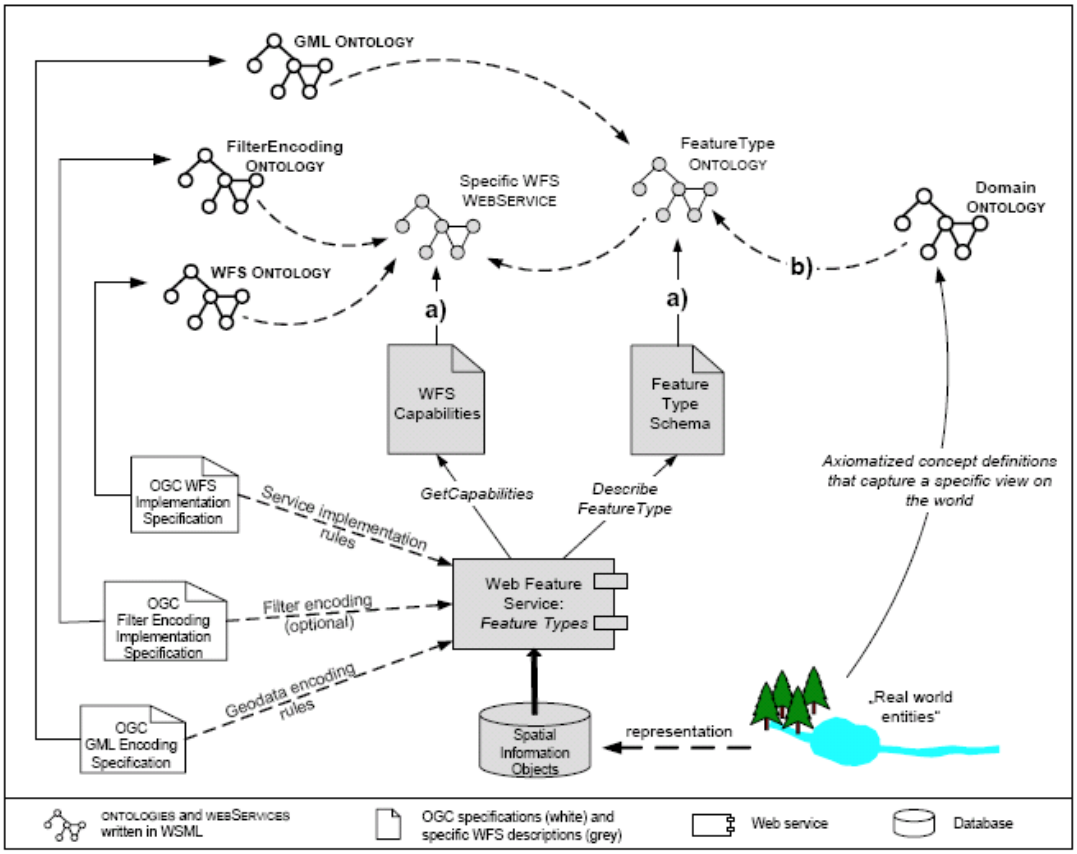



**Figure 23.** Information items involved in the annotation of a WFS in WSMX (Klien (2007), modified).

### 5.7.2 Requirements for WEBSERVICE specification for Geoprocessing Services

So far, most approaches in the context of geospatial Web Services that use formalized domain knowledge for annotation solely deal with geographic data services. Processing services are different to data providing services in the respect that their functional descriptions have to specify the service's input and its relation to the output, which makes them much more complex (Fitzner and Hoffmann, 2007). For example, since distances can be calculated on almost every two objects that have a spatial attribute, it is not sufficient for a service offering distance calculations only to consider the output (the distance) in its functional description; the functional description also has to include the types of input that are accepted by a specific service.

In the context of the testbed, we can thus formulate the following requirements that functional descriptions (i.e. semantic annotations) of geo-processing services should fulfill:

- they have to consider the types of in- and output and also constraints on them (e.g. a service that calculates a distance between two spatial objects requires both objects to be in the same coordinate reference system)
- they have to consider the relation from input to output
- can be integrated into the underlying SWS-framework.

As illustrated in Figure 25, the description of the service functionality takes into account background ontologies on operations and geographic data-types, in order to ensure a consistent use of terminology as defined in the OGC specification. The typology of operations is based on a set of well-known atomic GIS operations (e.g. as specified in ISO 19107 Spatial Schema (ISO/TC211, 2003).

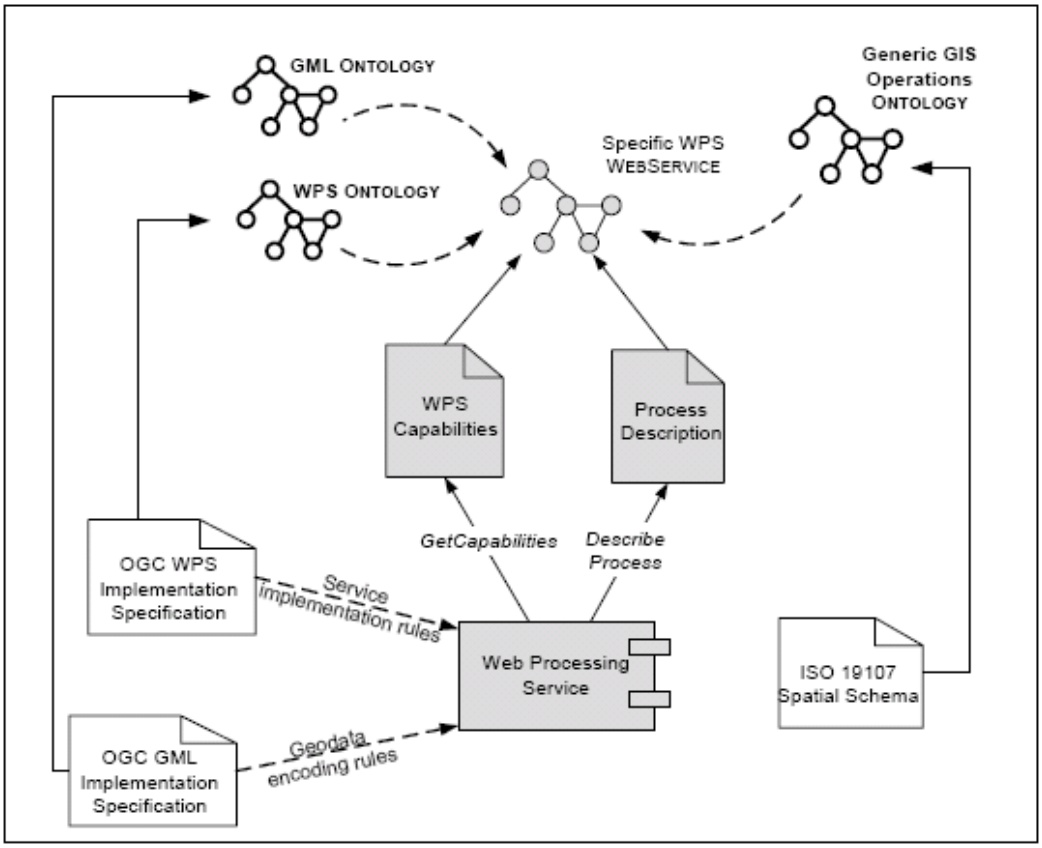



**Figure 24.** Information items involved in the annotation of a WPS in WSMX (in analogy to Figure 1).

Against this background, the generation of a WEBSERVICE for a WPS requires to use generic GIS operation types defined in the Domain Ontolog and to further constrain the parameters in the pre- and postconditions of the Web Service according to the specific WPS functionalities.

### 5.7.3 Semantic validation based on semantic annotation

The aim of validating a service composition semantically is to avoid unexpected behavior or wrong results at the thematic level. This kind of behavior is likely to appear, when a service in the composition either does have different semantics then the creator of the composition had in mind or a service is invoked with input data which has unsuitable semantics. The following section discusses validation mechanisms, which try to minimize the risk of creating such a faulty composition.

Semantic validation of service chains is useful for compositions containing web services offered by different organizations. Since the combination of different services for information exchange is one of the motivations to use workflows, this is a likely scenario. In contrast the workflow used in OWS-6 is uses services which were created with the purpose to fit for this special task. Therefore in this case semantic heterogeneities are very unlikely. We take another example for a composition, where semantic validation makes sense. We assume the combination of a WFS providing a road network with a WPS that calculates a noise distribution map. The creator of a workflow discovers a service which has the keyword infrastructure attached but is offering waterways. The WPS would syntactically also able to process waterways, which would produce a result which is semantically invalid. This can be avoided by using semantic service descriptions and would be identified as an error by the validation engine.

In contrast to web service discovery, the requirements regarding the semantic descriptions for semantic validation are slightly different. Fitzer and Hoffmann (2007) state that semantic discovery should be based on descriptions containing at least:

- type signatures
- constraints on input and output
- the operation that is performed
- the dependencies between input and output

The strategy for a semantic validation requires less information. The type signatures are not relevant since the syntactic issues are expected to be solved in a working service composition and there are a number of tools available to support the creation of compositions at syntactic level. Also, the performed operation is of no interest, since the validation aims at finding heterogeneities at the invocation of services. Of course the descriptions of the operations carry information that might be helpful to judge if the overall functionality of the service composition is valid, but this is not subject of the

validation at the moment. The same holds for the dependencies between input and output. At the time of the service invocation in a composition only the constraints on the input are of interest. These constraints specify the semantic annotations with concepts from some domain ontology. Therefore it is possible to use subsumption reasoning to judge if the annotated input satisfies the constraints on the input by the service. This approach on validation can be used to validate the waterway / road network example.

The example WFS delivers infrastructure data about waterways. Therefore output constraints for a semantic validation would contain the information shown in Listing 18. Namespaces are followed by the '#' character. Words, which begin with '?' are WSML variables, elements in '[]' indicate local attributes and words followed by parentheses global relations. The annotate relation is central, because it connects elements of the data model to a domain ontology. Following this, Listing 18. reads like: the WFS returns instances of the type 'WFSOutputType', which have the two attributes: 'hasGeometry' and 'hasTrafficDensity'. The annotation relations reveal that the second attribute provides information about the density of traffic on water ways.

```
1  ?x ofType [hasGeometry ofType ?y, hasTrafficDensity ofType ?s] fto#WFSOutputType and
2  ?z memberOf ont#WaterWays and
3  annotate(?x?z) and
4  ?t memberOf ont#TrafficDensity and
5  annotate(?s,?t) and
6  ont#hasDensity(?z,?t).
```

**Listing 18 WSML description of a WFS output.**

While the input constraints of the WPS that calculates the noise level look like Listing 19. As in the case of the WFS output, the input type has two attributes: 'hasGeometry' and 'hasTrafficDensity'. But this time, the density is linked to road networks. i.e. the 'hasTrafficDencity' attribute has another meaning.

```
1  ?x ofType [hasGeometry ofType ?y hasTrafficDensity ofType ?s] fto#WPSInputType and
2  ?z member of ont#RoadNetwork and
3  annotate(?y,?z) and and
4  ?t memberOf ont#TrafficDensity and
5  annotate(?s,?t) and
6  ont#hasDensity(?z,?t).
```

**Listing 19 WSML description of a WPS input.**

A subsumption reasoner, like IRIS (Vasiliu, 2006), which evaluates the containment query with these two expressions would clearly give a negative result and therefore the validation could inform the user that the input is not suitable.

This simple example illustrates the value of semantic validation of (geo-) processing workflows. When workflows become operational in heterogeneous environments,

semantic annotation is unavoidable. Support in discovery and validation is a core component of future SDI interoperability.

# 6 Scenario

The OWS-6 GPW scenario is constructed around an airport fire emergency case. First responders have to be coordinated with stakeholders from different domains. The workflow related concepts developed in this testbed and described in section 5 are applied to a specific part of the overall scenario:

The First Respondence Officer (FRDO) from the regional authority domain coordinates all actions. In order to determine, whether or not the plume generated by the fire will cross the runway or the entry lane, the plume has to be simulated with temporal and spatial constraints.

A workflow will be invoked by the FRDO which generates the desired plume simulation. The workflow is realized as an opaque workflow exposed as a WPS process and spans across security domains.

The following section provides a static overview of all participants followed by a dynamic view focusing on the interaction between different workflow partners and the message exchange against the background of the developed concepts in this testbed.

## 6.1 Architecture

The scenario involves several entities from three different security domains as depicted in figure 26.

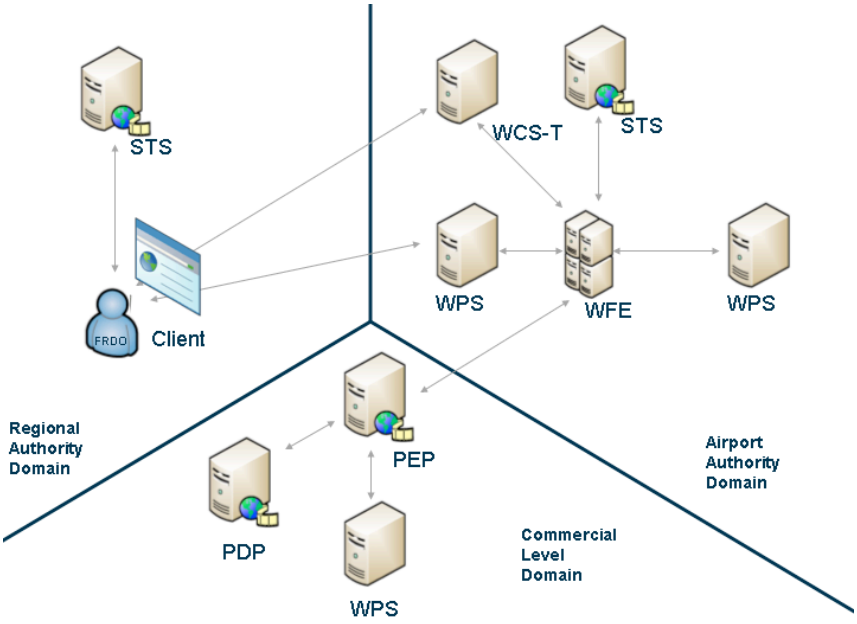



**Figure 25.** Scenario Architecture

### Regional Authority Domain

The previously mentioned FRDO resides in the regional authority domain. Figure 27 shows the simple interface the FRDO can use to invoke the workflow.

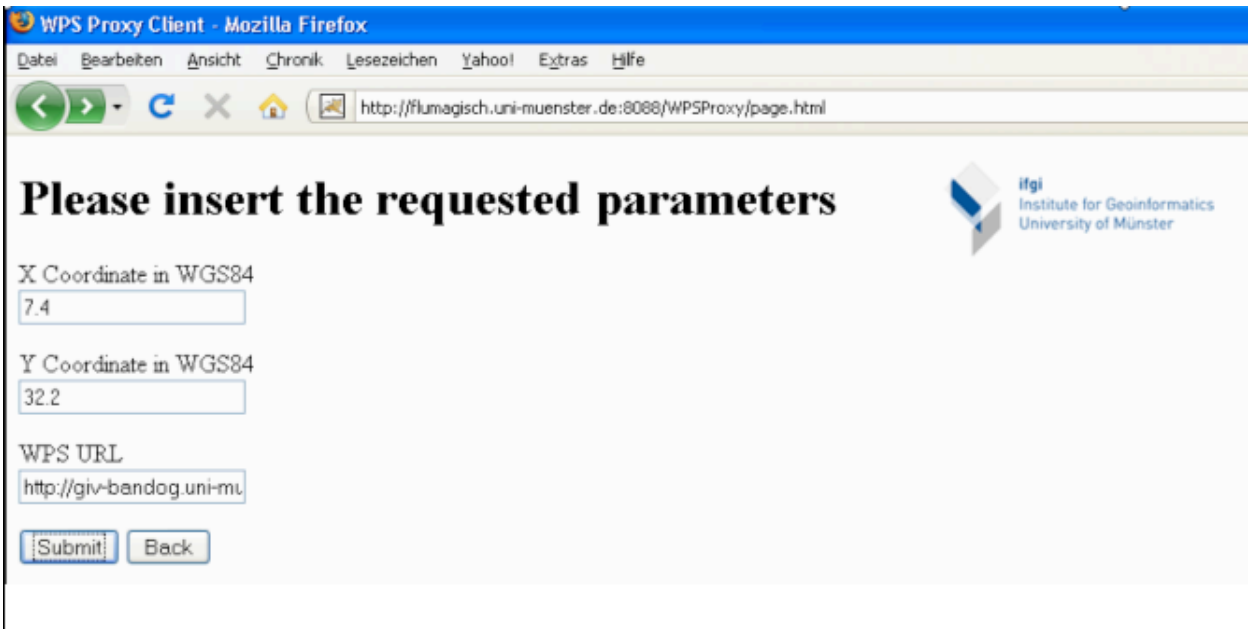



Figure 26. **FRDO interface**

**The X and Y coordinate have to be inserted along with the URL of the WPS that exposes the workflow as a WPS process. There, from the FRDO's perspective, the plume is generated by a simple WPS process. The complex internal logic is encapsulated.**

Besides the simple client, the regional authority domain contains a Security Token Service (STS) created by University of Muenster-IfGI. The STS interface is in general specified by WS-Trust. The service is responsible for issuing security tokens that could be understood by trusted parties.

**Airport Authority Domain**

The airport authority domain consists of several parties.

A Web Processing Service (WPS) created by University of Muenster-IfGI wraps the workflow with a WPS interface and therefore is the workflow front end.

A BPEL Workflow Engine created by GMU responsible for orchestrating the workflow.

A Security Token Service (STS) created by University of Muenster-IfGI in charge of issuing identity tokens for the Airport Authority Domain.

A Web Processing Service (WPS) created by UK Science and Technology Council calculates weather predictions.

A Transactional Web Coverage Service (WCS-T) created by GMU responsible for storing the workflow results as WCS coverages.

**Commercial Level Domain**

A Policy Enforcement Point (PEP) created by ConTerra/IfGI enforces access rights for the secured plume model WPS.

A Policy Decision Point (PDP) created by ConTerra/IfGI decides whether or not a requestor has access to the secured service.

A Web Processing Service (WPS) created by University of Muenster-IfGI calculates a plume model based on weather predictions.

### 6.2 Dynamic Model

The dynamic model focuses on the interaction between the workflow participants. Figure 28 visualizes the interaction. Requests are orange, responses green.

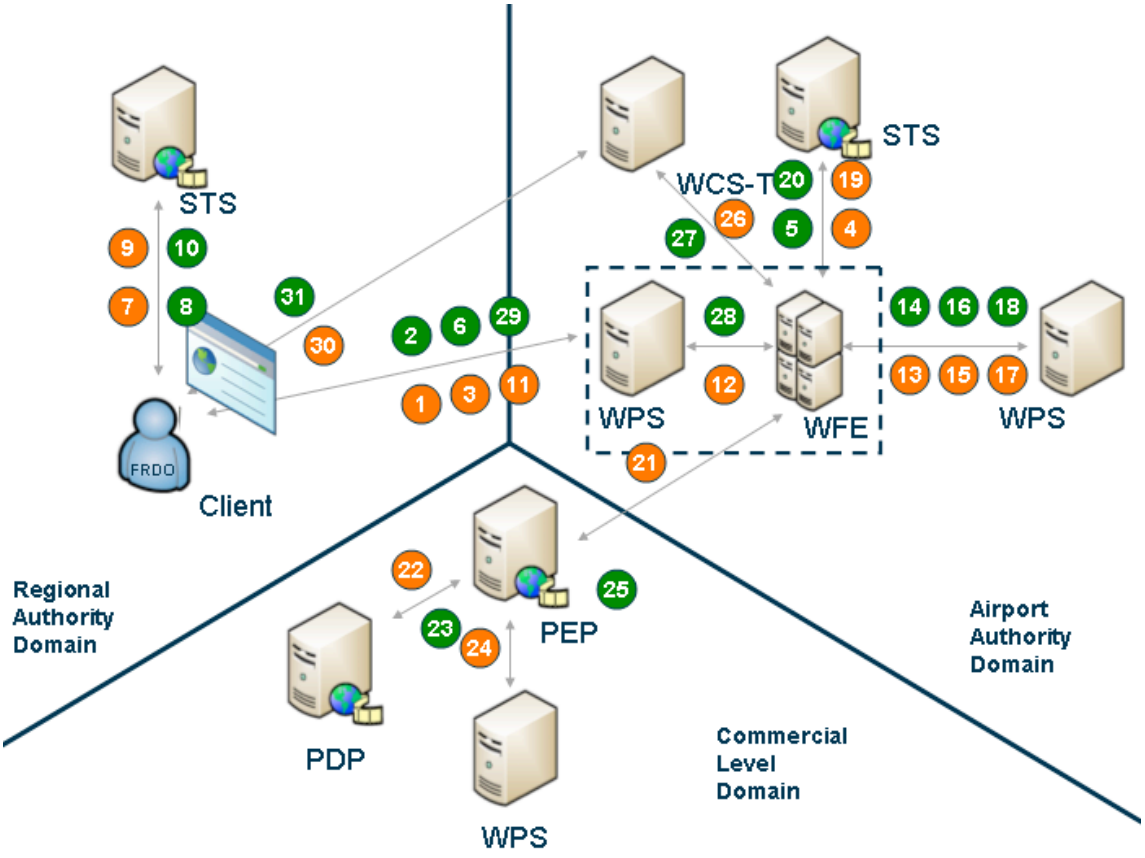



**Figure 27. Scenario Interaction Model**

At first the FRDO enters the coordinates of the fire incident and initiates the workflow in order to generate the desired plume simulation using the simple client interface shown in figure 27.

Internally the simple client initiates the workflow: In Transaction 1 (1), the service metadata from the workflow wrapped as a WPS process is requested via the GetCapabilities operation. In (2), the metadata is responded back and is analyzed to identify any preconditions. In our case, the preconditions are attached to the GetCapabilities Response as described in section C and encoded as WS-Policy. In the next step, the reference URL is extracted and since the client understands WS-Policy, the preconditions are obtained from the WPS workflow wrapper in (3).

According to section 5.6, an Identity Token is needed to be attached to the WS-Policy document. Therefore, in (4) an identity token is requested from the Airport Authority domain's STS and delivered in (5). The WS-Policy document with embedded identity token is returned back to the client in (6).

The WS-Policy response is analyzed and noted that a delegation token is required to invoke the workflow. A delegation token will allow the workflow engine to act on the client's behalf for dedicated purposes.

Therefore, two steps are required:

First: An identity token for the client is requested from the Regional Authority domain's STS (7)(8).

Second: The client's identity token as the *delegator* and the workflow engines identity token (extracted from the WS-Policy) as the *delegate* are send to the STS to retrieve a delegation Token (9)(10)

After all required tokens are gathered, the client sends an execute request to the workflow wrapper(11).

The operator now sees that the workflow is invoked and processing as shown in figure 29.

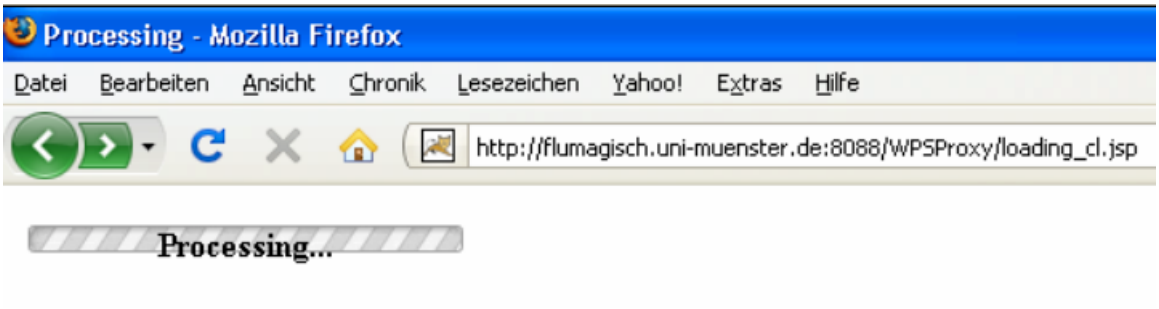



**Figure 28.** The workflow was initiated

The WPS workflow wrapper checks if all required token are present and invokes the BPEL workflow engine including the security tokens (12).

In the next step, the first workflow partner (Trajectory WPS) is invoked asynchronously (13). In this case, the pull model is used which lets the workflow engine directly response with a status document (14) (see Annex X for full documents).

The workflow engine is now obliged to periodically request (15)(17) the trajectory WPS (intermediate response (16))until the process has finished (18). When the requested process indicates that the process results are available, the workflow engine has to extract the URL provided from the trajectory service in the process finalization message

representing the process results. With this URL, the workflow engine can obtain (19) the simulated plume data from the trajectory WPS (20).

In the next step, the workflow engine has to take the results from the trajectory service and use them as input for the plume service WPS. Since the plume service is secured, first an identity token from the domain's STS has to be obtained (21) (22) in order to authenticate at the workflow participant.

This Secured Plume WPS is also invoked asynchronously via the WS-Addressing Push Mechanism described in section 5. (23). Since the workflow engine has no previously established trust relationship nor has any rights stored in the security system of the plume service WPS, the delegation token has to be attached to the request and used for this purpose. Besides, the request is digitally signed for non repudiation purposes.

The policy enforcement point (PEP) proofs if the requestor has provided all required security tokens and formulates a request to the policy decision point (24).

The policy decision point (PDP) recognizes that the workflow engine is not allowed to request the secured WPS. But it also recognizes that the client (which is allowed to request the secured WPS) has issued a delegation token which delegates the access rights on a one time use basis to the workflow engine. It also recognized that the delegation token was not issued by the client itself, but by a STS which was also authorized by the client to issue the token. Therefore, the policy decision point allows the unknown workflow engine to access the secured WPS (25).

Upon the positive decision by the PDP, the policy enforcement point forwards the request to the plume WPS (26). The WPS calculates the plume prediction and returns it back to the call back address which is the workflow engine (27).

After receiving the process results, the workflow engine can now dynamically store the process results in a Web Coverage Service via the transactional interface (WCS-T) (28). The resulting GetCoverage request (29) to the newly store WCS Layer is then delivered back to the WPS wrapper (30).

And the WPS wrapper delivers the GetCoverage URL back to the client (31) which is then presented to the FRDO as shown in figure 30.

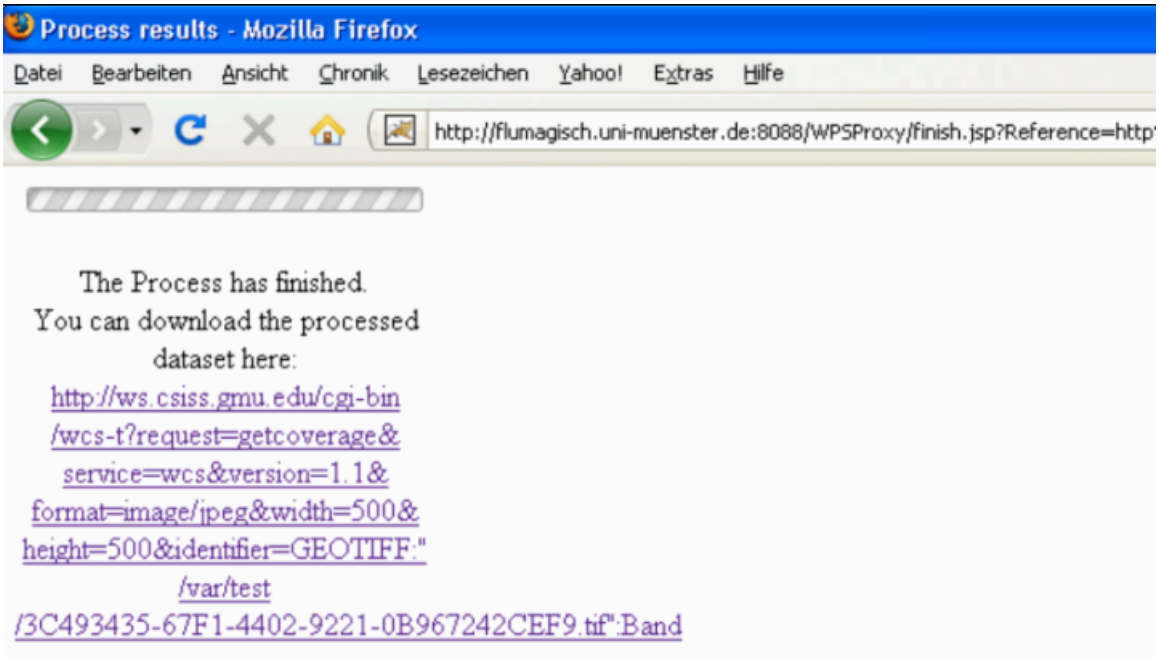



**Figure 29. Workflow Result reference as WCS coverage**

The FRDO is then able to extract the GetCoverage URL representing the workflow results from the WPS response and can fetch the data from the WCS (32)(33).

The resulting WCS coverage (figure 30) can then be used for further analysis, e.g. overlay it with the airport geometry.

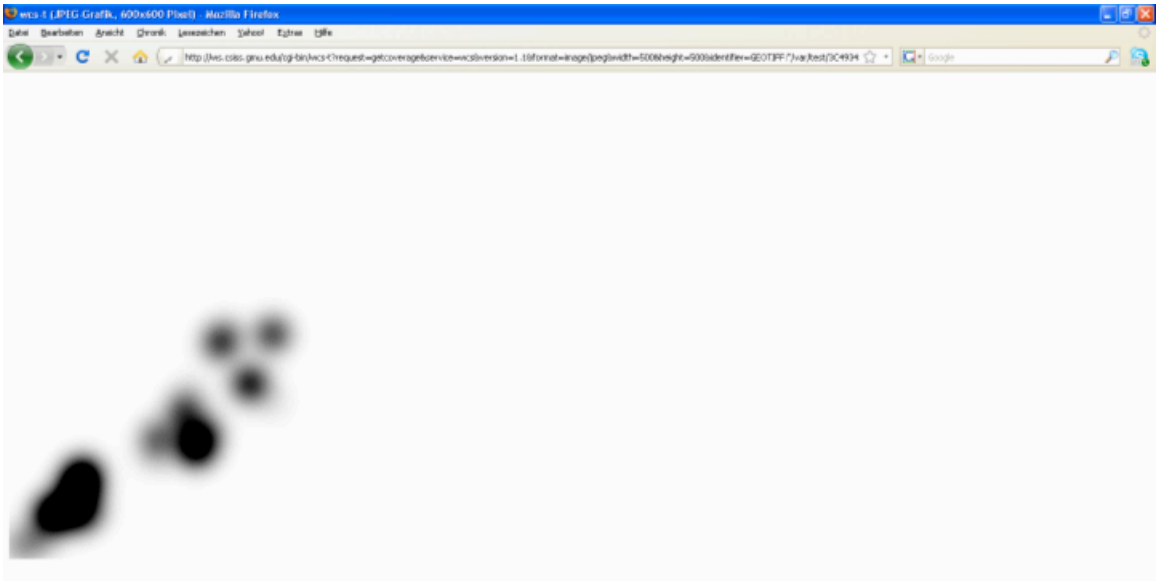


**Figure 30.** Visualized workflow results fetched from a WCS.

# 7 Summary

This OWS-6 Geoprocessing Workflow Architecture Engineering Report presented several methods for Geoprocessing Workflows in a SOA environment. Starting with a definition of the term Geoprocessing Workflow, going over different methods of exposing GPW in a standardized way up to data patterns. Besides security aspects and in particular the delegation of authority aspects, the ER showed also how to model workflows in a language independent fashion and utilize asynchronous methods and as well as how to enrich workflows with semantic aspects.

The Fire Threat Scenario served as a successful proof of concept implementation in order to demonstrate the applicability of the various concepts developed in this testbed.

Even thought many aspects have been covered by this ER, there are still some open questions in the GPW arena:

- How to use hybrid workflow with SOA and ROA workflow participants?
- How to map security aspects from SOA to ROA and vice versa?
- How to advance the security concepts with licensing in workflows?
- How to automatically assemble workflows?
- How to bring Human Interaction in a workflow?
- How does cloud computing relates to GPW?
- How to include semantics in standard service descriptions?
- How to use semantics in standard discovery and in data retrieval?
- How to make semantic validation operational? What is the required support infrastructure?
- ...

## References

Alameh N. (2003): Chaining geographic information Web Services. IEEE Internet Computing, Sept-Oct 2003, pp. 22-29.

Andrei, M., A. Berre, L. Costa, P. Duchesne, D. Fitzner, M. Grcar, J. Hoffmann, E. Klien, J. Langlois, A. Limyr, P. Maue, S. Schade, N. Steinmetz, F. Tertre, L. Vasiliu, R. Zaharia, and N. Zastavni (2008). SWING: An Integrated Environment for Geospatial Semantic Web Services. Demonstration Paper at the European Semantic Web Conference (ESWC'08).

Andrews, T., Cubera, F., Dholakia, H., Goland, Y., Klein, J., Leymann, F., Liu, K., Roller, D. Smith, D., Thatte, S., Trickovic, I. and Veerawarana. S. (2003): Business process execution language for Web Services version 1.1., OASIS, Online: :http://download.boulder.ibm.com/ibmdl/pub/software/dw/specs/ws-bpel/ws-bpel.pdf.

BAE Systems, IBM, Microsoft, SAP (2004). Web Services Metadata Exchange

Bussler, C., E. Cimpian, D. Fensel and J.M. Gomez (2005). Web Service Execution Environment (WSMX). W3C Member Submission. http://www.w3.org/Submission/WSMX/, [last accessed 15/04/2009].

Chen, L., Wassermann, B., Emmerich, W., Foster, H (2006): Web Service Orchestration with BPEL. London Software Systems; Dept. of Computer Science, University College London. Online: http://sse.cs.ucl.ac.uk/omii-bpel/publications/tut15-emmerich.pdf.

de Bruijn, J., H. Lausen, et al. (2006). The Web Service Modelling Language WSML: An Overview. Berlin and Heidelberg, Springer.

Duchesne, P., P. Maué, et al. (2008). "Semantic annotations in OGC standards - OGC Discussion Paper." Retrieved 15.04., 2009, from http://portal.opengeospatial.org/files/?artifact_id=31102&version=1.

Gasser, M.and McDermott, E. (1990) "An architecture for practical delegation in a distributed system," in IEEE Computer Society Symposium on Research in Security and Privacy, 1990. Proceedings, pp. 20-30, 1990.

Gone, M. and S. Schade (2008): Towards Semantic Composition of Geospatial Web Services – Using WSMO in Comparison to BPEL, International Journal of Spatial Data Infrastructures Research (IJSDIR), Vol 3.

Grcar, M. and E. Klien. (2007). Using Term-matching Algorithms for the Annotation of Geo-services. In Proceedings of Web Mining 2.0 Workshop. In conjunction with ECML-PKDD 2007, Warsaw, Poland.

Guarino, N. (1998). Formal Ontology and Information Systems. Formal Ontology in Information Systems. N. Guarino. Amsterdam, IOS Press: 3-18. Fitzner, D. and J. Hoffmann : Functional Description of Geoprocessing Services as Conjunctive Queries . In Geographic Information Science Days Workshop 2007 (GI-Days 2007).

Hoff, H., A. Limyr, H. Midelfart, D. Skogan, and A.J. Berre (2006). SWING: D6.1 The Architecture of the Development Environment. Project Deliverable.

Hoffmann, J., N. Steinmetz and D. Fitzner. (2008). SWING: D2.4 Semantic Web Geoprocessing Services. Project Deliverable.

ISO 19119 (2001): *Geographic Information - Services*, ISO TC211 document number N1203.

ISO/TC211 (2003). 19107 Geographic information - Spatial Schema. ISO/TC211 Standards. ISO/TC211.

Kanneganti, R., Chodavarapu, P. (2008). SOA Security. Manning

Kiehle C., Greve K., and Heier C. (2006): Standardized Geoprocessing - Taking Spatial Data Infrastructures one Step Further.In: 9th AGILE Conference on Geographic Information Science,Visegrad, Hungary, pp273-282.

Klien, E. (2007). "A Rule-Based Strategy for the Semantic Annotation of Geodata." Transactions in GIS, Special Issue on the Geospatial Semantic Web 11(3).

Klien, E., S. Schade, and J. Hoffmann (2007). SWING: D3.1 Ontologies in the SWING Application. Project Deliverable.

Lembo, A, (2004): Illustrating Classic GIS Tasks, Cornell University, Online: http://dspace.library.cornell.edu/handle/1813/165?mode=full

Leymann, F., Roller, D., Thatte, S. (2003): Goals of the BPEL4WS Specification. Working document submitted to the OASIS Web Services Business Process Execution Language, OASIS, Online: http://xml.coverpages.org/BPEL4WS-DesignGoals.pdf .

OGC (2003). Web Notification Service. Discussion Paper. OGC# 03-

OGC (2004). OWS-2 IH4DS Service Chaining IPR. OGC #04-078

OGC (2006). OWS-4 Workflow IPR. OGC #06-178r1

OGC (2007). OGC Web Services Common Specification version 1.1.0 with Corrigendum 1. OGC# 06-121r3

OGC (2007). Web Processing Service. OGC Implementation Specification. OGC# 05-007r7

OGC (2008) Transactional Web Processing Service. Discussion Paper. OGC# 08-123

OGC (2009) OWS-6 SWE Event Architecture Engineering Report. OGC# 09-032

OGC (2009) OWS-6 Security Engineering Report. OGC# 09-035

Russell, N., ter Hofstede, A.H.M. ,Edmond, D., and van der Aalst, W.M.P. (2005). Workflow Data Patterns. QUT Technical report, FIT-TR-2004-01, Queensland University of Technology, Brisbane

Schade, S., Klien, E., Maué, P., Fitzner, D., and Kuhn, W. (2008). SWING: D3.2 Report on Modelling Approach and Guideline. SWING Project.

Scicluna, J., Polleres, A., and Roman, D. (2006). Ontology-based Choreography and Orchestration of WSMO Services, at http://www.wsmo.org/TR/d14/v0.2/, [last accessed 15/04/2009].

Stollberg, B. (2006): Geoprocessing in Spatial Data Infrastructures - Design and Implementation of a Service for Aggregating Spatial Data. Master Thesis. FH Mainz. (2006)

Vasiliu, L. (2006). SWING: D2.2 Web Service Execution Environment Integrated with a Reasoning Engine. SWING Project Deliverable.

van der Aalst, W.M.P (2003): Don't go with the flow: Web services composition standards exposed. In: IEEE Intelligent Systems, 01/02 2003, pp.72-76, Online: Electronically accessible from http://www.tm.tue.nl/it/research/patterns/ieeewebflow.pdf

W3C (2003). Web Services Adressing. Online: http://www.w3.org/Submission/2004/SUBM-ws-addressing-20040810/

Welch, V., Foster, I., Kesselman, C., Mulmo, O., Pearlman, L., Tuecke, S., Gawor, S. Meder, S. and Siebenlist, F. (2004) X. 509 proxy certificates for dynamic delegation, in 3rd Annual PKI R&D Workshop, 2004,

Wang, J., Del Vecchio, D. and M. Humphrey (2005) Extending the security assertion markup language to support delegation for web services and grid services, in ICWS 2005. Proceedings. 2005 IEEE International Conference on Web Services, 2005., pp. 67-74.

Weiser, A., Neis, P. and Zipf, A. (2006): Orchestrierung von OGC Web Diensten im Katastrophenmanagement - am Beispiel eines Emergency Route Service auf Basis der OpenLS Spezifikation. In: GIS - Zeitschrift für Geoinformatik, pp.c35-41.

Wohed, P., van der Aalst, W.M.P., Dumas, M. and ter Hofstede, A.H.M. (2003): Analysis of Web Services Composition Languages: The Case of BPEL4WS. In: ER, LNCS 2813, Springer-Verlag Berlin Heidelberg, pp.200–215.

Zhang, L., Ahn, G. J. and Chu, B. T. (2003) "A rule-based framework for role-based delegation and revocation," ACM Transactions on Information and System Security (TISSEC), vol. 6, 2003, pp. 404-441.