BLEKINGE INSTITUTE OF TECHNOLOGY
School of Computer Science

MASTER THESIS

Automation of data processing in the network of geospatial web services

AUTHOR:
Dawid Nejman

SUPERVISORS:
Ph.D Tomasz Kubik
Ph.D Stefan Johansson

WROCŁAW, 2013
Contents

List of shortcuts iv

List of figures vi

List of tables vii

1. Introduction 8
   1.1. Aims and objectives ........................................... 11
   1.2. Research questions ............................................ 11
   1.3. Goal and scope .................................................. 12
   1.4. Research methods ............................................... 12
   1.5. Content description ............................................ 12

2. Background 14
   2.1. W3C web services ............................................... 14
       2.1.1. Standards, protocols, interfaces .......................... 15
       2.1.2. Service orchestration and choreography .................. 19
   2.2. Geospatial web services ........................................ 21
       2.2.1. Standards, protocols, interfaces .......................... 22
       2.2.2. Service orchestration ...................................... 29
   2.3. Workflows ....................................................... 30
       2.3.1. Languages ................................................... 32
## List of shortcuts

<table>
<thead>
<tr>
<th>Acronym</th>
<th>Full Form</th>
</tr>
</thead>
<tbody>
<tr>
<td>API</td>
<td>Application Programming Interface</td>
</tr>
<tr>
<td>BioVel</td>
<td>Biodiversity Virtual e-Laboratory</td>
</tr>
<tr>
<td>B2B</td>
<td>Business to Business</td>
</tr>
<tr>
<td>CC-BY-SA</td>
<td>Creative Commons Attribution Share-Alike</td>
</tr>
<tr>
<td>CPS</td>
<td>Coverage Portrayal Service</td>
</tr>
<tr>
<td>CSW</td>
<td>Catalog Service for the Web</td>
</tr>
<tr>
<td>ebRIM</td>
<td>ebXML Registry Information Model</td>
</tr>
<tr>
<td>ebXML</td>
<td>electronic business using eXtensible Markup Language</td>
</tr>
<tr>
<td>FTP</td>
<td>File Transfer Protocol</td>
</tr>
<tr>
<td>GIS</td>
<td>Geographic Information System</td>
</tr>
<tr>
<td>GML</td>
<td>Geography Markup Language</td>
</tr>
<tr>
<td>HTTP</td>
<td>Hypertext Transfer Protocol</td>
</tr>
<tr>
<td>ISO</td>
<td>International Standardization Organisation</td>
</tr>
<tr>
<td>JSON</td>
<td>JavaScript Object Notation</td>
</tr>
<tr>
<td>KVP</td>
<td>Key-Value Pair</td>
</tr>
<tr>
<td>MTOM</td>
<td>Message Transmission Optimization Mechanism</td>
</tr>
<tr>
<td>OASIS</td>
<td>Organization for the Advancement of Structured Information Standards</td>
</tr>
<tr>
<td>OdbL</td>
<td>Open Database License</td>
</tr>
<tr>
<td>OGC</td>
<td>Open Geospatial Consortium</td>
</tr>
<tr>
<td>OWS</td>
<td>OGC Web Service Common</td>
</tr>
<tr>
<td>QoS</td>
<td>Quality of Service</td>
</tr>
<tr>
<td>Acronym</td>
<td>Description</td>
</tr>
<tr>
<td>---------</td>
<td>-------------</td>
</tr>
<tr>
<td>REST</td>
<td>Representational State Transfer</td>
</tr>
<tr>
<td>SCUFL</td>
<td>Simple Conceptual Unified Flow language</td>
</tr>
<tr>
<td>SDI</td>
<td>Spatial Data Infrastructure</td>
</tr>
<tr>
<td>SOA</td>
<td>Service Oriented Architecture</td>
</tr>
<tr>
<td>SOAP</td>
<td>Simple Object Access Protocol</td>
</tr>
<tr>
<td>SOS</td>
<td>Sensor Observation Service</td>
</tr>
<tr>
<td>SVG</td>
<td>Scalable Vector Graphics</td>
</tr>
<tr>
<td>SWE</td>
<td>Sensor Web Enablement</td>
</tr>
<tr>
<td>UDDI</td>
<td>Universal Description Discovery and Integration</td>
</tr>
<tr>
<td>URL</td>
<td>Universal Resource Locator</td>
</tr>
<tr>
<td>WCS</td>
<td>Web Coverage Service</td>
</tr>
<tr>
<td>WFS</td>
<td>Web Feature Service</td>
</tr>
<tr>
<td>WMS</td>
<td>Web Map Service</td>
</tr>
<tr>
<td>WPS</td>
<td>Web Processing Service</td>
</tr>
<tr>
<td>WPS-T</td>
<td>Transactional Web Processing Service</td>
</tr>
<tr>
<td>WS-BPEL</td>
<td>Web Service-Business Process Execution Language</td>
</tr>
<tr>
<td>WSCI</td>
<td>Web Service Choreography Interface</td>
</tr>
<tr>
<td>WSDL</td>
<td>Web Services Description Language</td>
</tr>
<tr>
<td>W3C</td>
<td>World Wide Web Consortium</td>
</tr>
<tr>
<td>XML</td>
<td>Extensible Markup Language</td>
</tr>
<tr>
<td>XOP</td>
<td>XML-binary Optimized Packaging</td>
</tr>
</tbody>
</table>
## List of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1</td>
<td>Structure of SOAP message</td>
<td>17</td>
</tr>
<tr>
<td>2.2</td>
<td>Web service sequence</td>
<td>19</td>
</tr>
<tr>
<td>2.3</td>
<td>Web service split</td>
<td>20</td>
</tr>
<tr>
<td>2.4</td>
<td>Web service join</td>
<td>20</td>
</tr>
<tr>
<td>2.5</td>
<td>Web service iteration</td>
<td>20</td>
</tr>
<tr>
<td>2.6</td>
<td>Orchestration and choreography</td>
<td>21</td>
</tr>
<tr>
<td>2.7</td>
<td>OGC service chaining</td>
<td>25</td>
</tr>
<tr>
<td>2.8</td>
<td>CSW-ebRIM specification dependencies</td>
<td>28</td>
</tr>
<tr>
<td>2.9</td>
<td>BPEL process definition</td>
<td>31</td>
</tr>
<tr>
<td>2.10</td>
<td>Taverna architecture</td>
<td>34</td>
</tr>
<tr>
<td>3.1</td>
<td>Client-coordinated service chaining</td>
<td>35</td>
</tr>
<tr>
<td>3.2</td>
<td>Static service chaining</td>
<td>36</td>
</tr>
<tr>
<td>3.3</td>
<td>Workflow-managed service chaining</td>
<td>37</td>
</tr>
<tr>
<td>4.1</td>
<td>Taverna workbench</td>
<td>42</td>
</tr>
<tr>
<td>4.2</td>
<td>Structure of single execution block</td>
<td>43</td>
</tr>
<tr>
<td>4.3</td>
<td>SOS service invocation plugin package diagram</td>
<td>45</td>
</tr>
<tr>
<td>4.4</td>
<td>SOS service discovery plugin package diagram</td>
<td>46</td>
</tr>
<tr>
<td>4.5</td>
<td>Configuration dialog</td>
<td>46</td>
</tr>
<tr>
<td>4.6</td>
<td>Processor details</td>
<td>47</td>
</tr>
<tr>
<td>4.7</td>
<td>Update sites window</td>
<td>50</td>
</tr>
<tr>
<td>4.8</td>
<td>Updates and plugins window</td>
<td>50</td>
</tr>
<tr>
<td>4.9</td>
<td>Adding plugin site</td>
<td>51</td>
</tr>
<tr>
<td>4.10</td>
<td>Workflow returning waterlevel from SOS plugins</td>
<td>51</td>
</tr>
<tr>
<td>4.11</td>
<td>Workflow returning map from WMS plugins</td>
<td>52</td>
</tr>
<tr>
<td>4.12</td>
<td>Map result from WMS processor</td>
<td>53</td>
</tr>
<tr>
<td>4.13</td>
<td>Full workflow</td>
<td>54</td>
</tr>
</tbody>
</table>
List of Tables

2.1. Web service standard stack ...................................................... 16
2.2. Geospatial web service standard stack ........................................ 23
2.3. Web Map Service mandatory operations ....................................... 25
2.4. Selected Web Feature Service operations ..................................... 26
2.5. Selected Web Coverage Service operations .................................... 27
2.6. Core Sensor Observation Service operations .................................. 27
2.7. Selected Web Feature Service operations ..................................... 29
2.8. BPEL and SCUFL comparison .................................................... 33

3.1. Transactional Web Processing Service operations ............................. 38
Chapter 1

Introduction

Geographic science plays substantial role in everyday life on Earth. The climate change as well as recent catastrophes make it even more relevant than before. Nowadays, computer science is used to utilize vast amounts of geospatial data. It is used in areas like weather prediction, wildlife monitoring, water and land resource management, disaster management and so on. Furthermore, processed information gives more insight and understanding into still unknown phenomena.

Government agencies have the biggest influence on distribution of geographical information [1]. Partially it derives from the fact, that they are the main provider of geographic information but also because they can control national development through laws and policies. Governments around the world have started to consider a change in geospatial data distribution in the advent of Geospatial Information System (GIS) technology.

According to definition, GIS is a system designed to digitally capture, manipulate, analyze, merge and present geographical data. Its potential was spotted back in 60’s, but not until late 80’s were the first steps taken to distribute geospatial data in that way. This was the time when first generation of spatial data infrastructures (SDI) were proposed across many developed countries including Australia, Canada (Canadian Geospatial Data Infrastructure), Japan (Japanese National Spatial Data Infrastructure), United Kingdom (British National Geospatial Data Framework), United States (National Digital Geospatial Data Framework) and others. The SDI, according to Nebert is “often used to denote the relevant base collection of technologies, policies and instrumental agreements that facilitate the availability of and access to spatial data.” [2].

An accurate case that argued the need for national SDI, was made at the beginning of President Clinton’s Executive Order for the US National Spatial Data Infrastructure from 1994:

“Geographic information is critical to promote economic development, improve our stewardship of natural resources and to protect the environment. Modern technology now permits improved acquisition, distribution, and utilization of
geographic (or geospatial) data and mapping. The National Performance Review has recommended that the Executive Branch develop, in cooperation with State, local and tribal governments and the private sector, a coordinated National Spatial Data Infrastructure to support public and private sector applications of geospatial data in such areas as transportation, community development, agriculture, emergency response, environmental management and information technology.” [3]

The effects of this order, as well as other efforts that have followed, are clearly visible in modern life. As predicted, the SDI initiatives started all over the world not only expanded scientific knowledge, but also stimulated economic growth. This created a chance for private sector to provide services based on distributed geographical data. Now, private companies are the second main source of geospatial data, usually exposed through proprietary application programming interfaces (API). There is another emerging supplier of geospatial data - community of Internet users organized in so-called social networks. One example of such social network is OpenStreetMap (OSM) which (as it says) is a project that creates and provides free geographic data and mapping. The purpose of the project is to give access to cartographic data without restrictions that private companies usually impose. What is important is the fact, that the rights to the data belongs to the contributors, who also decide under what terms of use it is being published.

It is worth to notice, that as important as political decisions are technical aspects of geoscience. Main problems here are the amount of data needed to be processed, methods of distribution, data conversion and so on. In other words, computer systems dealing with geospatial information had to handle data distributed across many other systems in many different formats and technologies. Those obstacles have been overcome by the use of web service technology [4].

According to Tsalgatidou “a web service is a self-describing, self-contained, modular application accessible over the web” [5]. Because technologies behind web services are standardized, many different vendors provide their own implementations that abide to the standard. Consequently, web services can share data and process control, regardless of the platform and technologies they use [4]. This allows distributing computation alongside the data across different modules over the network. Moreover, it is also possible to use remote resources (hardware, software, information) published as a web service.

The rising popularity of web services technology, gave birth to Service Oriented Architecture (SOA). Organization for the Advancement of Structured Information Standards

It is important to note that the OSM data is also licensed. This means that not everything is allowed. Currently OSM maps are under Open Database License (OdbL) 1.0. This license was imposed recently and replaced another license - Creative Commons Attribution Share-Alike 2.0 (CC-BY-SA) which is not recommended for informational databases. As stated on the website, a “Share-Alike” license is better than “Public Domain” license for several reasons. One of them is the possibility that other companies would be able to use OSM data without making it free for use.
(OASIS) defines SOA as “a paradigm for organizing and utilizing distributed capabilities that may be under the control of different ownership domains” [6]. This means that SOA is a design methodology utilizing business functions that can be controlled and maintained by different entities. Furthermore, SOA promotes reuse and interoperability of web services, allowing to get more value from capabilities that are local in addition to those under the control to others [6]. This is why, SOA became so important for the development of distributed computer systems and all enterprise application.

Web services with underlying paradigms provide more possibilities for data handling and distribution compared to traditional ways of data dissemination [4]. This was the reason behind development of geospatial web services, which are a combination of web service and geospatial information. Nowadays many geospatial web services exist, deployed by different organizations and offering data at different level of details. Their strength derives from the compliance to standards and, what comes next, their interoperability. This is evident especially in the context of the Reichardt’s note: “no single organization produces all the data (...), and no single vendor provides all the systems” [7]. Furthermore, the ability to solve more complex geospatial tasks by composing several different services is one of the greatest values of geospatial web services [8].

Services composition is usually done through choreography and orchestration. Both terms describe methods used to create business processes from web services. Although their meaning overlaps in some respect, they mostly point to different aspects of creation process. “Orchestration refers to an executable business process that can interact both with internal and external web services” [9]. Communication between separate services is done at a message level. Messages describe logic and task execution. Orchestration “always represents control from one party’s perspective” [9]. This means that one central unit (service) is responsible for orchestrating process execution between other web services. Consequently, orchestrated web services are not aware about the business process they participate in. This is the main difference between orchestration and choreography, which in turn “allows each involved party to describe its part in interaction” [9]. Choreography describes public message exchange between different web services. Both orchestration and choreography are implemented by the use of standards created for web services.

There are differences between “standard” web services and theirs geospatial variant. They come from the fact, that different organizations were involved in its creation. One of them is created by collaboration of World Wide Web Consortium (W3C) and OASIS. Another entity dealing mostly with geospatial science is Open Geospatial Consortium (OGC). This resulted in different set of standards used across separate sets of services. While web services used in business can be easily composed to work together, the same process is more cumbersome for geospatial science.
1.1. **Aims and objectives**

The aim of this paper is to address the automation in geospatial data processing. This will be achieved by fulfilling following objectives:

- To identify current achievements in geospatial web service orchestration,
- To check the possible options of chaining / orchestrating geospatial web services,
- To design and implement proof-of-concept of OGC web services orchestration,
- To evaluate results with alternative approaches.

The outcomes of the objectives will help to answer the research questions.

1.2. **Research questions**

In order to direct the efforts of this thesis, following research questions (RQ) have been identified. Answers to those questions will help to identify and fill the research gap.

RQ1. What have been already done to automate data processing in geospatial web services?

RQ2. What are the methods to make geospatial web services chaining / orchestration more dynamic?

RQ3. How effective is the new approach proposed in this thesis?

Starting from the first research question, this thesis will try to provide references to work that have been made in the field of geospatial web services orchestration. The question will be answered by completing the first objective: “to identify current achievements in geospatial web service orchestration”. This will also specify research gap and direction of the thesis.

Second research question seeks to list methods that automate data processing in geospatial web services available at the time of writing. This record will not be comprehensive but rather selective, including most recent methods and attempts. Next step and consequently an extension of second research question will try to explore further one chosen method. This part was included in third objective: “to design and implement proof-of-concept in a form of web application”.

Finally, third research question will try to evaluate the outcomes of this thesis. To answer this question, thesis will fulfill the last objective: “to verify results with alternative approaches”.

11
1.3. Goal and scope

As mentioned before, the main goal of this master thesis is to propose a way to automate data processing in geospatial web services. The automation should be understood as an attempt to orchestrate or at least chain web services used in geospatial science. The goal will be further elaborated in Chapter 4, where the problem description is provided.

The scope of this thesis covers problem description, development of proof-of-concept and evaluation of the outcomes. It would be wrong to suggest that this thesis tries to give a universal solution that will orchestrate all kinds of geospatial web services. Instead, it is merely an attempt that should show the direction for future research.

1.4. Research methods

This paper makes use of several different research methods: literature search, experiment and results validation. The choice of these methods is a direct outcome of research questions defined before. Each and every one of them is best suited to provide an answer for one of the research question.

Literature search was performed in order to gain knowledge about general use of web services, current methods of service orchestration / chaining and past attempts to chain / orchestrate geospatial web services. The outcomes are described in Chapter 2 (Background).

Experiment “involves an investigation of causal relationships using tests controlled by yourself” [10]. Furthermore it is often used for development, evaluation and problem-solving projects [10]. This is why the main part of the thesis was done in form of an experiment. It consists of three stages: design, implementation and evaluation of the outcomes. All of them are covered in Chapter 4 (Experiment).

1.5. Content description

This thesis has been divided into following sections:

Chapter 1 (Introduction) - provides quick introduction to the topic of geoscience, web services, and composition methods. Next, the chapter defines four objectives and resulting research questions. Connections between objectives, research questions are defined. Last but not least, goal and scope of thesis are defined. Finally, Chapter 1.4 introduces research methods that are used to answer research questions.

Chapter 2 (Background) - presents exhaustive background to the topics used across this paper. Chapter breaks information into two types of web services: W3C / OASIS and OGC. The reason for differences between these two types of services is explained.
The most important part of this chapter is devoted to technologies and standards that are connected to different kind of web services. As important as the stack of standards is workflow part, where concepts like orchestration, chaining, workflow languages and engines are explained. Final part of the chapter is about Taverna engine used in this paper.

Chapter 3 (Theory) - provides discussion on existing ideas associated with the problem. Provides latest attempts to solve the problem.

Chapter 4 (Experiment) - presents methods used (or adapted) to solve the described problem. Contains description of techniques used to orchestrate geospatial web services. Chapter starts with requirements analysis based on described objectives and then goes into design of the architecture. Final part of this chapter is devoted to implementation, tests and evaluation of the outcomes.

Chapter 5 (Discussion) - starts with self-critical discussion of method used to perform master thesis.

Chapter 6 (Summary) - final part of the master thesis. It summarizes outcomes and provides direction for future research.
Chapter 2

Background

In order to understand the nature of problem tackled in this thesis, there is a need for a background to the technologies and standards that are being used. Chapter will start with the main unit of business functionality - web service. Definitions included will be helpful to explain more complex structures - business processes described by workflows. Finally, this chapter will end with a part on orchestration engine.

2.1. W3C web services

As defined by W3C, web service is a “software system designed to support interoperable machine-to-machine interaction over a network. It has an interface described in a machine-processable format (...). Other systems interact with the Web service in a manner prescribed by its description (...).” [11]. In other words, web service can be understood as an application exposed over the Internet that can communicate with other applications by using a set of standardized protocols. The first company that introduced the term “web service” in connection with technologies like XML, SOAP, WSDL and UDDI was Microsoft in 2000 [12].

Probably, the most important feature of web services is the ability to create loosely coupled and reusable software components [13]. This has been achieved by the set of standards that were defined by both W3C and OASIS consortia. The standards, with the technology they use, help to hide the implementation details behind the common interface. When talking about W3C / OASIS web services, the interface is described in machine-processable format using Web Services Description Language (WSDL). The description tells other applications, what services are being provided and how to execute them. To interact with the web service, other applications usually use Simple Object Access Protocol (SOAP) messages transferred by HTTP protocol with Extensible Markup Language (XML) serialization [11].

Based on interaction model there are two main categories of web services: arbitrary and
REST-compliant web services [14]. Arbitrary web services are also referred as “message-oriented” services. This is because the basic units of communication between different services are SOAP messages. Arbitrary web services are used to build systems according to SOA, which was explained in the introduction. On the other side there is Representational State Transfer (REST) architecture which constrains operations on web services to a set of standard HTTP commands (e.g., GET, POST, PUT, and DELETE). This is done in order to “manipulate XML representation of Web resources using a uniform set of stateless operations.” [14]. Worth noting is the fact, that SOAP (starting from version 1.2) can be used in manner that is consistent with REST. In that case the interface can be built on top of SOAP using for instance WS-Transfer (Table 2.1) technology.

Architecture presented above is of considerable significance for business, because it releases from complex and slow software integration [13]. Instead companies can focus on the actual functionality of their services. The medium for service distribution and communication is the Internet itself. This helps with building complex Business to Business (B2B) solutions, based on functionality exposed in web services.

2.1.1. Standards, protocols, interfaces

Due to general interest in SOA in enterprise environment over the years, there have been many technologies that were added to the standard. Those technologies can be divided into several main categories depending on the role they pose. One of the possible division was presented in Table 2.1. Bold text in table marks core technologies that are addressed below. This chapter will also describe the few most important technologies that form a core functionality of web services. In addition to simple description, chapter will try to give an insight into the current state of each technology.

First level of standard stack is communication layer, which is also equivalent to last OSI model layer - application layer. At this level there are defined protocols and methods responsible for communication. One of them, which is of particular interest, is Hypertext Transfer Protocol (HTTP) which is a foundation of data communication for World Wide Web. Therefore, it is used for communication between web services. Standard can be used in two ways - directly or indirectly. Web services based on REST architecture expose theirs API as a set of HTTP request methods. Such API can be accessed by direct HTTP calls. The result is usually an XML or JSON document. Since not all web services architectures’ are REST-based, some use SOAP messages for communication. This is especially common for so called “big web services” which provide a way for both Remote Procedure Call (RPC) and messaging [15]. Such services use HTTP to transfer XML messages that obey SOAP standard.

Moving one level higher from communication layer there is data layer. The core standard for data exchange and serialization with connection to web services is XML. Due to its structure, XML is highly flexible and extensible. What is important about XML
<table>
<thead>
<tr>
<th>Process</th>
<th>BPEL, SCUFL, WS-CDL, WSCI</th>
<th>Tab. 2.1. Web service standard stack [4, 14].</th>
</tr>
</thead>
<tbody>
<tr>
<td>Orchestration</td>
<td>WS-Transactions, WS-Coordination, WS-CAF</td>
<td></td>
</tr>
<tr>
<td>Transaction</td>
<td>ASAP</td>
<td></td>
</tr>
<tr>
<td>Asynchronous service</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Metadata</td>
<td>WSDL*, OWL-S</td>
<td></td>
</tr>
<tr>
<td>Service description</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Cataloguing</td>
<td>UDDI, WSIL</td>
<td></td>
</tr>
<tr>
<td>Policy</td>
<td>WS-Policy, WS-PolicyAssertions</td>
<td></td>
</tr>
<tr>
<td>Other</td>
<td>WS-MetadataExchange</td>
<td></td>
</tr>
<tr>
<td>Messaging</td>
<td>WS-Event, WS-Notification</td>
<td></td>
</tr>
<tr>
<td>Events</td>
<td>WS-Enumeration, WS-Transfer</td>
<td></td>
</tr>
<tr>
<td>Sessions</td>
<td>WS-Addressing, WS-MessageDelivery</td>
<td></td>
</tr>
<tr>
<td>Routing</td>
<td>WS-Reliability</td>
<td></td>
</tr>
<tr>
<td>Reliability</td>
<td>SOAP, MTOM</td>
<td></td>
</tr>
<tr>
<td>Message</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Type</td>
<td>XSD, DTD, OWL</td>
<td></td>
</tr>
<tr>
<td>Schema</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Data</td>
<td>SVG</td>
<td></td>
</tr>
<tr>
<td>Vector</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Data</td>
<td>XML, JSON</td>
<td></td>
</tr>
<tr>
<td>Communications</td>
<td>HTTP, SMTP, FTP, JMS, IIOP</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

language to web services is the architecture of the language: XML Infoset, XML Schema and XML Namespace [14]. XML Infoset is a “set of information items and their associated properties that comprise an abstract description of an XML document” [14]. Another element is XML Schema. It is a description of XML document, containing definitions of types and constrains. It helps to create new XML grammars. One example of such grammar is SOAP, which itself is defined by XML Schemas. Another important property of XML Schema is the ability to validate XML documents against schemas that describes them. This is very convenient for applications that use XML documents because, after XML validation application can expect data that is formatted according to schema. What is more, there are libraries that can apply this in two ways. Not only is it possible to validate XML documents but also to create valid documents from data provided based on XML Schema. Finally, there is XML Namespace, which allows using elements coming
from different grammars across one XML document.

Although, web service technologies are based on standards this does not mean they are perfect. Over the years some new, more promising approaches have emerged. In 2007, Zhao et al. as a conclusion wrote that “compared to other distributed computing approaches, Web service is relatively poor in performance(...)” [4]. The cause of this, comes from many factors. One of them is slow data serialization and distribution. XML as a universal and extensible language, impose a lot of unnecessary data. This throttles communication between services. RESTful web services allowed for other serialization methods. One of them is JSON. Despite the fact that more standards decrease interoperability, as Pautasso et al. have suggested “the benefit of preferring JSON over XML can outweigh the extra effort and the lack of interoperability with a significant overhead reduction” [15]. JSON is a text-based open standard derived from JavaScript language. It is mainly used for serializing data structures and transmitting them over the network.

Another level on the stack is messaging. The main messaging protocol used by web services is SOAP. It is a XML syntax (or to be more specific a XML Infoset) which defines messaging architecture and format. In other words it is an “envelope” packaging XML message. Such messages can be serialized in many ways, one of which is XML document. Other example can be Message Transmission Optimization Mechanism (MTOM) - a method of sending binary data between web services using XML-binary Optimized Packaging (XOP). This serialization method is used in cases when there is a need for smaller message size. Regardless of serialization method, SOAP message consists of the following elements: envelope, header and body (Figure 2.1). The envelope is a top-level XML element that wraps entire message and holds message header and body. The SOAP header is an extension mechanism that contains information about routing, Quality of Service (QoS) and configuration. It allows a SOAP message to be extended for various uses of SOAP. Finally SOAP body contains application-specific data, described by proprietary XML Schemas. One of the SOAP strengths is protocol independence. The same message can be sent through a variety of different systems using many transport protocols [15]. An example of SOAP message is presented in Listing 2.1. This is a message in protocol version 1.2 making a plane ticket reservation.

![SOAP Message Diagram](image-url)

**Fig. 2.1.** Structure of SOAP message.
On top of messaging layer stands metadata layer. There are two technologies that are usually mentioned in connection with “big” web services: Web Service Description Language (WSDL) and Universal Description, Discovery and Integration (UDDI). WSDL is a language used to describes messages and operations that are available on specific web service. It also provides a syntax to describe specific information regarding protocols used on web service and the format of messages that are accepted. WSDL is an XML-based language and has its own XML Schema to validate the content of created files. These files have “.wsdl” as an extension. The convenience of WSDL technology is in its transparency [2]. WSDL document can be generated automatically, based on the interface that is being published. Furthermore, web service described by WSDL enables programs called “proxy generators” to automatically construct a request for that web service. As it is described below, transparency makes orchestration and chaining easier to implement [2].

Finally, the last of the “Big Four” standards (along with XML, WSDL and SOAP) released in 2000 is UDDI. This standard in theory was meant to serve as a “yellow pages” for web services. The idea was that companies would publish their web services to the UDDI registry which in turn would allow automatic discovery of relevant services for other applications. This did not catch up as expected. There were many factors that decided against this technology. One of them can be the fact that most current UDDI models are centralized [16]. This has negative influence on performance when registry is handling many web services. Another drawback is lack of automatic mechanism for updating the registry [16]. The result of this is that the information on web services is not up to date with recent changes. Finally, many of the systems simply did not need a web service registry to handle metadata and service discovery for few web services. Due to lack of interest companies like IBM, Microsoft or SAP have closed their public UDDI nodes in
January 2006. One year later OASIS UDDI Specification Technical Committee voted to complete its work.

2.1.2. Service orchestration and choreography

The final layer of standard stack, both for regular, as well as geospatial web services is process layer. This layer defines standards and technologies that operates on web services. Operations that use services as a building blocks are referred as business processes. Depending on the way that different web services are interconnected we can identify different methods of business processes implementation: orchestration and choreography.

As it was described before in the introduction, both orchestration and choreography are methods for creating business processes [9]. Zhao et al. provides more specific definition of orchestration. He states that orchestration is a process of “assembling individual geospatial Web services into chain for representing a more complicated geospatial model and process flow” [4]. Despite clear reference to geospatial web services, this definition can be treated as universal. Processes can be chained in several ways. Usually process chain includes structures like: sequence (Figure 2.2), split (Figure 2.3), join (Figure 2.4) and iteration (Figure 2.5).

All structures are controlled by central unit. This is the main difference between orchestration and choreography. The latter tracks messages that are being exchanged between web services. Contrary to orchestration, in choreography every service plays its part. Because choreography is collaboration between services, it only describes tasks that involve communication between them. Furthermore, the choreography does not include information on the steps (tasks) that service provider performs internally. It is only interested in the provided service [17]. This is another difference between the two concepts. Orchestration describes all the structural information about internal tasks that need to be

Sequence

![Sequence Diagram](image)

Fig. 2.2. Example of web service sequence. Following elements are:

1 - history of prices,
2 - inflation calculation service,
3 - history of inflation,
4 - interest rate calculation service,
5 - real interest rate.
executed after service invocation. The difference between orchestration and choreography is presented in Figure 2.6.
2.2. Geospatial web services

After description of W3C web services it is time to talk about its geospatial cousins and concentrate on the differences between them. Geospatial web service can be described as modular application that provides services, information and knowledge on geospatial data [4]. Like other services, geospatial web services have a set of standard operations during its lifetime: publication, discovery, binding, invoking and execution. Despite the similarities, OGC web services are different from the web services based on W3C and OASIS standards. Standards for geospatial web services were developed in parallel to those from W3C and OASIS. This is the main source of differences between two types of services. As a good example can serve WSDL, UDDI and SOAP technologies which still are at the core of standard web services and are at most optional for geospatial web services.

The lack of WSDL standard (or to be more precise, the optional implementation for some of OGC interfaces) is one of the more substantial differences between the two types
of services. This is because, OGC decided to provide an interface directly through HTTP protocol. Services with such interface are now regarded as RESTful (based on REST architecture). There was a strong argument for such move [18]. One of the problems with WSDL adaptation is binary-only response to some requests. Since there is no “xml:binary” type, it is not possible to send raw data through XML. Furthermore it would make no sense to send binary data in that way. This, as well as other reasons make geospatial web services different from standard web services. The key challenge that they face - the need to support for very large data sets, force them to make appropriate architecture choices [19].

In order to publish geospatial data OGC has developed many standards across variety of fields. OGC Specifications define interface methods that web service implementation needs to support. For instance, there are two specifications: Web Feature Service (WFS) and Web Coverage Service (WCS). They do not have methods for data visualization. Alternatively, they publish methods with raw data output. This data can be later retrieve by other web service, processed into pictures and published through another OGC interface like Web Map Service (WMS). As defined, WMS “produces maps of spatially referenced data dynamically from geographic information” [20]. This must be obeyed by every WMS implementation. Web Map Service implementation can transform data coming from specified source like database management system, WFS or WCS\(^1\). For geocomputation functions OGC designed Web Processing Service (WPS). It is an interface that “facilitates the publishing of geospatial processes, and the discovery of and binding to those processes by clients” [21]. More complete picture of standards and technologies connected to geospatial web services is presented in Table 2.2.

### 2.2.1. Standards, protocols, interfaces

Similarly to the W3C, OGC geospatial web services use a set of standards and technologies that can be presented as a stack (Table 2.2). Although, the layers of the stack are similar to those of W3C services, first element that is noticeable is the lack of orchestration layer. This is because there are no clearly designated technologies that can be used to interconnect geospatial web services without any work needed. This problem will be described in greater detail later on.

As noted before, the main technology of communication layer is HTTP. This is also true for standard web services based on REST architecture. It is worth noting that OGC would like to further integrate its geospatial web services with REST by implementing GeoServices REST API [23]. It is a candidate standard submitted by four organizations:

---

\(^1\)From that point on, OGC web service specification will be treated equally to implementation of that service. For instance, Web Processing Service will be equal to geospatial web service implementation compliant to WPS specification. This is done in order to make following definitions and examples easier to understand.
2.2. GEOSPATIAL WEB SERVICES

<table>
<thead>
<tr>
<th>Process</th>
<th>Query</th>
<th>ISO 19125-1</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Integrative</td>
<td>WPS</td>
</tr>
<tr>
<td>Metadata</td>
<td>Service description</td>
<td>WSDL, ISO 19119, ISO 19109</td>
</tr>
<tr>
<td></td>
<td>Data description</td>
<td>ISO 19115:2003, ISO/TS 19139</td>
</tr>
<tr>
<td></td>
<td>Cataloging</td>
<td>CSW, UDDI, WSIL, ebRIM, ISO metadata</td>
</tr>
<tr>
<td>Messaging</td>
<td>Application interfaces</td>
<td>WCS, WMS, WFS, WICS, WCTS, SOS</td>
</tr>
<tr>
<td></td>
<td>User interface</td>
<td>WMC</td>
</tr>
<tr>
<td></td>
<td>Message</td>
<td>HTML GET / POST, SOAP</td>
</tr>
<tr>
<td>Type</td>
<td>Schema</td>
<td>XSD, DTD, OWL</td>
</tr>
<tr>
<td>Data</td>
<td>Vector</td>
<td>OGC-GML, SensorML, OGC-WKT/WKB</td>
</tr>
<tr>
<td></td>
<td>Data file</td>
<td>SDTS, VPF, DIGEST, HDFEOS</td>
</tr>
<tr>
<td></td>
<td>Data</td>
<td>XML, ASCII, JSON, GeoJSON</td>
</tr>
<tr>
<td>Communications</td>
<td></td>
<td>HTTP, SSL, SMTP, FTP, IIOP</td>
</tr>
</tbody>
</table>

Tab. 2.2. Geospatial web service standard stack [4, 22].

Ersi Inc., interactive instruments GmbH, Oracle USA and 52°North. Its main purpose is to convert specification made by Ersi Inc. and widely used in the business environment for over four years into OGC standard. By using this API clients could request resources residing on the server by using Universal Resource Locators (URLs). As an answer, client (i.e. web browser or web service) would get map images, text location information or other data. Moreover, communication with this API is JSON-based so the data overhead is smaller compared to standard SOAP solutions. Integration of GeoServices REST API is a move that is beneficial for all parties involved. OGC benefits in staying on top of popular and widely used technologies. On the other hand Ersi Inc. ensures that its products are compliant to OGC standards.

What sets geospatial web services apart is the data layer. Output from geospatial web services can be either in text or binary format. Text data is usually an XML document described by appropriate grammar. One such grammar is Geography Markup Language (GML) which is defined by standard as an “XML encoding (...) for the transport and storage of geographic information modeled in accordance with the conceptual modeling
framework (...) and including both the spatial and non-spatial properties of geographic features.” [24]. It means that it can be used both as a modeling language for geographic systems as well as format for interchanging geographic data over the Internet. GML contains description of geometry objects like point, line and polygon. Using those three objects it can describe physical structures like roads, borders, rivers and others. Starting from version 3.0, GML includes also raster data such as various images and sensor data. There are many other XML-based languages which made into OGC standard. For example there is SensorML standard which can be used to describe sensors and measurement data. Another example is CityGML - language for storing and exchanging virtual 3D city models.

As mentioned before, JSON became more popular for sending geospatial data due to low data overhead. This is especially true with the new GeoServices REST API. There is another format that is directly based on JSON - GeoJSON. This is an open format for encoding geographic data structures like: point, line, polygon, and others. It is a collaborative project that aims to use JSON format for geographic data encoding. Since its start in 2008, the project has become quite popular (e.g. Twitter uses GeoJSON to send geotagging parameters). It is also easy to implement. Since GeoJSON document is compliant with JSON structure, any JSON tools can used to process GeoJSON data. Nevertheless, despite its clear advantages, GeoJSON has not been included to the GeoServices REST API proposal.

Although, most text data is send using XML serialization, there are varieties of other formats that can be used as an output: compressed data (gzip), images (gif, png) and others. Output format depends on the type of service and method used. For general communication, textual data is preferred. Binary output on the other hand is used to retrieve data that is too big, or cannot be send in text format. In some cases it is possible to request an output format. For instance, when client requests a map from Web Map Service (WMS), one of the parameters determines output image format.

Another on stack is messaging layer. As it has been said before, HTTP protocol is the main way to access geospatial web services. For instance, the proposed GeoServices REST API uses exclusively HTTP POST and GET methods [23]. On top of that are build web service interfaces. There are many types of geospatial services. Each of them defines its purpose and also indicate which methods are available in its interface.

One such type is Web Map Service (WMS) which, as noted before, produces spatial data from geographic information. WMS interface defines two mandatory operations presented in Table 2.3.

The data is in the map form that can be rendered in several different formats, mostly pictorial (PNG, GIF or JPEG) or vector (SVG). Requests are submitted in form of the URL. In this way using appropriate HTTP GET method it is possible to indicate which part and what kind of the map service should provide. Furthermore, because output for-
Tab. 2.3. Web Map Service mandatory operations.

<table>
<thead>
<tr>
<th>Operation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GetCapabilities</td>
<td>operation that returns machine and human-readable description of service. Contains information about service type, version etc. This operation is mandatory across most of OGC web services.</td>
</tr>
<tr>
<td>GetMap</td>
<td>operation returns map that satisfy given mandatory and optional parameters (like layer, bounding box, width, height, format etc.).</td>
</tr>
</tbody>
</table>

mats allow transparent backgrounds (e.g. GIF or PNG) it is possible to download different kind of maps with the same coordinates and put them together to make a composite map. The most obvious example can be weather map with Earth as a base layer and meteorological data on top. There are no restrictions in connection with service location. It is possible to download different maps from separate servers. This gives the possibility to use a network of distributed map servers to create customized maps [20]. A real life example of such scenario is presented in Figure 2.7. Here, the user in order to get a map connects to WMS from a web browser. This WMS (here created by NASA) in order to build a map combines maps from other WMS service with data from Coverage Portrayal Service (CPS). On the other side, CPS uses data from Web Coverage Services (WCS). Finally, both WCS and WMS are connected to databases containing geospatial data.

Another important type of geospatial web services is Web Feature Service (WFS). WFS is a standard that defines interface for feature access, coordinate and geographic format conversion. Standard is an important part of geospatial web service family because it has defined a new way of geographic data exchange on the Internet. As the standard points out “rather than sharing geographic information at the file level using File Transfer Protocol (FTP), for example, the WFS offers direct fine-grained access to geographic information

Fig. 2.7. Example of OGC service chaining [22].
at the feature and feature property level” [25]. Web services working with WFS are able to request specific information they need rather than implementing additional logic for data discovery and extraction. WFS can be accessed through two standard ways: HTTP GET / POST or SOAP over HTTP POST. This is in accordance with what has been said before. The item of transaction is usually an object called feature. It is a representation of word phenomena (like roads, rivers, borders etc.) in form of points and lines. Each feature has its own unique identifier and is encoded using GML format. WFS interface allows to use operations that discover, query, lock and store features. Additionally there is an option to perform operations in transaction. Some mandatory operations that WFS implements are listed in Table 2.4.

WMS and WFS return data that describe static phenomena. For data that contains information about space / time varying phenomena there is another OGC standard - Web Coverage Service (WCS). WCS operates on objects called coverages. Coverage is a “feature that acts as a function to return values from its range for any direct position within its spatiotemporal domain” [26]. Unlike WMS, WCS returns data with detailed description. This adds additional meaning to the data, which in turn can not only be displayed, but also interpreted, predicted, etc. WCS can also be interpreted as WFS service with additional dimension added to it. In that sense, coverage can be seen as a specialized type of feature with sometimes multidimensional range of properties describing

<table>
<thead>
<tr>
<th>Operation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GetCapabilities</td>
<td>discovery operation generating metadata about WFS service.</td>
</tr>
<tr>
<td>DescribeFeatureType</td>
<td>discovery operation describing what formats are used for feature input and output.</td>
</tr>
<tr>
<td>GetPropertyValue</td>
<td>query operation that outputs a set of features for a given value. Value can be a property of a single or complex feature.</td>
</tr>
<tr>
<td>GetFeature</td>
<td>query operation returns set of features that satisfy given query expression. Features are encoded with GML.</td>
</tr>
<tr>
<td>GetFeatureWithLock</td>
<td>query and locking operation returns set of features matching query expressions and locks them.</td>
</tr>
<tr>
<td>LockFeature</td>
<td>locking operation that ensures atomicity of feature update, so that other clients would not get feature that is in the middle of change.</td>
</tr>
</tbody>
</table>
space / time varying phenomena. Coverage model is based on GML Application Schema for Coverages. This ensure interoperability with other OGC Web Services (OWS). Most important operations that can be performed on WCS service are described in Table 2.5.

Last type of geospatial web service that will be discussed in this layer (but by no means the last OGC standard) is Sensor Observation Service (SOS). This service is a part of framework called Sensor Web Enablement (SWE). SWE provides a way to make sensors, transducers and sensor data repositories available via the Internet through many standards that it includes. As defined in the standard, SWE “aims at providing interfaces and protocols for enabling “Sensor Webs” through which applications and services are able to access sensors of all types” [27]. A sensor can be for instance a traffic monitor, web camera, environmental monitor or any other sensor that is connected to the net. Sensor Observation Service “defines a Web service interface which allows querying observations, sensor metadata, as well as representations of observed features” [27]. Furthermore, the interface allows to register or delete sensors. SOS service provides three core operations described in Table 2.6. There are other optional operations that are implemented as an extensions to SOS core layer. They include operations like: get observation by id, insert / delete sensor, insert result / observation, get result etc.

Another layer in the stack is metadata layer. Here, OGC uses Catalogue Service for the

Tab. 2.5. Selected Web Coverage Service operations.

<table>
<thead>
<tr>
<th>Operation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GetCapabilities</td>
<td>operation gives information about server’s capabilities as well as offered coverages.</td>
</tr>
<tr>
<td>DescribeCoverage</td>
<td>operation returns metadata on selected coverage.</td>
</tr>
<tr>
<td>GetCoverage</td>
<td>query operation that returns coverage accordingly to specified range properties at selected spatio-temporal location in given format.</td>
</tr>
</tbody>
</table>

Tab. 2.6. Core Sensor Observation Service operations.

<table>
<thead>
<tr>
<th>Operation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GetCapabilities</td>
<td>operation provides access to information about SOS service and other operations that the service makes available.</td>
</tr>
<tr>
<td>DescribeSensor</td>
<td>query operation retrieving information about sensors accessible through the service.</td>
</tr>
<tr>
<td>GetObservation</td>
<td>query operation retrieving observations. Those can be filtered spatially, temporally and thematically.</td>
</tr>
</tbody>
</table>
Web (CSW) Specification. CSW in turn specifies design patterns that allow for definition of interfaces called application profiles. Application profiles enable publish and search descriptive data. Profiles are based on standards such as mentioned before UDDI. In case of geospatial web services in use are two different profiles: ebRIM and ISO metadata. The ebRIM profile of CSW is in short referred to as CSW-ebRIM. As the standard defines CSW-ebRIM is a “specialized catalogue service based on the CSW part (Clause 10) of the OGC Catalogue Services 2.0.2 specification. In essence it joins the OGC CSW interfaces to the OASIS ebXML registry information model (ebRIM 3.0)” [28]. This specification structure is presented in a Figure 2.8. Each rectangle represents different specification. Rectangles are connected with arrows that can be translated into “depends on” relation. Finally all rectangles are grouped in packages based on its author. It may also be helpful to make a distinction between registry and catalogue. A registry is a special kind of catalogue with formal registration process (such as described in ISO 19135 or ISO 11179-6 standards). There is usually registration authority involved in enforcing use of proper procedures for accessing and maintaining registry content [28].

As portrayed in Figure 2.8, CSW-ebRIM directly depends on ebRIM, CSW, and SOAP. ebRIM is a model specifying the way in which different elements in catalogue are structured and connected. It is similar to UDDI but offers more flexibility and has more extensibility points. This suited OGC technology environment so it was chosen in 2007 as a preferred model foundation. While ebRIM provides the requirements to support the registration of services, the CSW defines requirements for data discovery and search - the main functions of cataloguing service. It is worth to mention that in parallel to ebRIM profile there is ISO metadata profile which is also used to publish and discover geospatial services.

Finally, similarly to the W3C web services, the last layer on the stack is process layer.

![Fig. 2.8. Dependencies of OGC CSW-ebRIM specification [28].](image-url)
What is different though is the lack of orchestration-related technologies. This has been pointed out by Zhao in his comprehensive geospatial technology overview. Over the time few implementations emerged which were trying to bring W3C-style orchestration using BPEL or other methods with various outcomes. This will be discussed in more details in following chapters. What is worth to mention in connection with process layer is a Web Processing Service (WPS) standard. If configured, WPS can provide any type of GIS functionality from basic calculations to complicated modelling solutions. Data needed by WPS can be retrieved from other services or from server. Because of its generic interface, service can wrap other OWS services. WPS interface operations are presented in Table 2.7. It is easy to see, that those are similar to the operations from services described before. What is interesting is the fact, that WPS is compatible with WSDL (which can optionally describe service interface). This is important part and the starting point for many attempts to orchestrate geospatial web services.

2.2.2. Service orchestration

In case of OpenGIS Web Services, orchestration is not as simple as it is for “regular” web services. There are several explanations for this, and one of them is the fact, that in some cases it is not possible to get service description in form of WSDL document. As mentioned before, WSDL is an “XML format for describing network services as a set of endpoints operating on messages containing either document-oriented or procedure-oriented information”\(^2\). The usual way to describe geospatial web service is to provide all information through GetCapabilities method. This method is obligatory for every OGC web service and provides information on service type, interface methods, input / output formats and others. Although, WSDL would be easier to implement and use, it is often not capable of describing specialist data services such as OGC web services [19].

This has been also proven in OGC examination of its standards (WCS, WFS, WMS, CSW) for compliance with WSDL / SOAP / UDDI technologies [18]. Several issues were

<table>
<thead>
<tr>
<th>GetCapabilities</th>
<th>operation provides names and descriptions of processes offered by WPS.</th>
</tr>
</thead>
<tbody>
<tr>
<td>DescribeProcess</td>
<td>query operation that returns detailed information about processes. This information also includes input / output requirements and data format.</td>
</tr>
<tr>
<td>Execute</td>
<td>operation executes implemented by WPS process.</td>
</tr>
</tbody>
</table>

\(^2\)http://www.w3.org/TR/wsd1
Complex types in HTTP bindings - data in WSDL is structured using Key-Value Pairs (KVP) bindings. Those pairs are then send using HTTP GET / POST methods. Each type must be matched to its SimpleType which prevents messages other than those with flat structure. Nevertheless, output messages can use complex types using available schemas to describe those types. Since some of the methods in OWS expects complex types on input this can be a source of problems.

Arbitrary parameters names - this problem is related to the way in which KVP are structured in WSDL document. Some OGC web services like Web Coverage Service (WCS) or Web Map Service publish methods that can take unrestrained number of parameters (like WMS GetMap or WCS GetCoverage methods). KVP in WSDL documents on the other hand must be exhaustive.

Binary response - as mentioned before, some OWS methods are using binary-type response (i.e. WMS GetMap or WCS GetCoverage). This proves to be an issue, because it does not make sense to send raw bytes alongside XML message and describe it by schema.

Asynchronous requests - this is the problem of CSW service because some of its requests may be processed asynchronously. For example, when accessing CSW GetRecords method, it returns an acknowledgement, echoing users request. Then service will process the requests as long as is needed and return list of records asynchronously. In spite of that, WSDL does only one output message to be defined.

Problems listed above are only related to WSDL description. There are other problems related to SOAP and UDDI. The reason for which they are important is because they pose additional effort to anyone trying to process more complex data using combined geospatial web services. This means it is not as easy as it is in case of W3C services to use them as building blocks. Certain amount of middleware is needed to implement, bind together and orchestrate geospatial services. Chapter 4 will cover in details how the orchestration part has been done and form a problem statement that will be explored in following parts.

2.3. Workflows

Central piece of orchestration is business process, which has been described as a “set of interrelated tasks linked to an activity that spans functional boundaries” [29]. Business process consists of two parts: process logic and routing rules. Process logic defines sequence in which different tasks should be executed. This property has been previously called
as chaining / orchestration (depending on the way in which sequence is made). The complementary part are routing rules between different tasks. Those two parts create a workflow, which can be described as “the way work gets from start to finish” [29]. Workflow helps to define who are the participants involved in given business process, what is there to be done and when to do it. A workflow participant can be one of the following types: a resource, human, system (automatic agent), organizational unit, etc. From the standpoint of this thesis, the emphasis is on web services. Therefore, the term of workflow participant will be narrowed to resources, which in turn can be web services, files, procedures, etc.

To execute a business process the process definition needs to be created first. It is usually represented in graphical form which is most suitable to represent elements of workflow and their relationships. Sample process definition is presented in Figure 2.9. An instance of the process definition is called job and is executed by workflow managed system. Workflow management system supply execution environment for running jobs. An example of Workflow management system can be Taverna, which will be covered in Section 2.4. Usually a client uses workflow managed system to define routing rules between process logic provided by developer.

The module responsible for executing jobs in workflow management system is workflow engine. It has three main functions:

- Verification: checking the status of tasks during the execution.
- Authorization: checking if user is allowed to execute given task.
- Execution: following tasks in defined in workflow.

![Fig. 2.9. Loan approval process as BPEL process definition.](image)

There are many different workflow engines. Each of them is compatible with different set of executable languages (more in Section 2.3.1) and resources.

### 2.3.1. Languages

In order to describe and later execute the workflow there is a need for a defined model. One of them is Web Service-Business Process Execution Language (WS-BPEL or BPEL). It is an “XML-based specification\(^4\) for describing the behavior of a business process that is composed of Web services and also exposed as a Web service” [30]. Despite the support for business workflows, it is attractive for scientific community due to support for SOA paradigm. Right now it is the most common language used to orchestrate web services implemented by companies like Apache Software Fundation, Microsoft, jBoss, Oracle Corporation, IBM and others [31].

A valid alternative for BPEL is Simple Conceptual Unified Flow Language (SCUFL) model. SCUFL is an XML document describing workflow. Similarly to BPEL, it is also a meta-model which defines how workflow can be described and executed. SCUFL treats web services like processors, which have input and output ports. Processor will run as soon as the data is available on the input port and will run in parallel by default if this is possible. Data is transported from processor to processor in XML wrapper despite the fact that it can take many formats. The processors are connected together by data links. On the other hand, execution order is enforced by control links every time when direct data flow is not required. In this case control link specifies that target processor cannot start until source processor has not finish it’s job. The main difference between SCUFL and BPEL is free flow of data and lack of specific control structures in SCUFL [31]. BPEL is more process oriented language and needs more explicit definition of control flow. Two models were very well compared by Missier et. al. (Table 2.8).

SCUFL model was mainly used with Taverna workflow management tool (Section 2.4). This has changed and it was replaced with t2flow model which additionally supported control and service extensions. Nevertheless, there were drawbacks to the move - t2flow model was using “heavy” XML serialization of Java beans. Another, more serious complaint, was coming from 3\(^{rd}\) party developers, who could not adapt so easily to the new format\(^5\) This was the reason to start works on new model: SCUFL2. The most important change is employing Linked Data technology to create platform-independent language that can be inspected, modified, created and executed.

---

\(^4\)There are few different meanings that are usually associated with BPEL. Most frequently BPEL means a type of language used to describe workflows, but sometimes it is (like in this thesis) referred as a meta-model.

\(^5\)More specifically, problem of adaptation came from the fact that the model was too tightly coupled with Java model. Source: [http://dev.mygrid.org.uk/wiki/display/developer/SCUFL2](http://dev.mygrid.org.uk/wiki/display/developer/SCUFL2).
### Tab. 2.8. BPEL and SCUFL comparison [30].

<table>
<thead>
<tr>
<th></th>
<th>BPEL</th>
<th>SCUFL</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Processing units</strong></td>
<td>activities</td>
<td>processors and processing units</td>
</tr>
<tr>
<td><strong>Meaning of links</strong></td>
<td>transfer of control</td>
<td>transfer of data</td>
</tr>
<tr>
<td><strong>Definition of data</strong></td>
<td>complex types initialized explicitly</td>
<td>automatic</td>
</tr>
<tr>
<td><strong>Control logic</strong></td>
<td>complete: sequence, conditional, parallel, event-triggered, etc.</td>
<td>limited: sequential, parallel and conditional</td>
</tr>
<tr>
<td><strong>Parallel execution</strong></td>
<td>defined in &lt;flow&gt; or &lt;ForEach&gt;</td>
<td>by default</td>
</tr>
</tbody>
</table>

### 2.4. Taverna

As mentioned before, Taverna is an open-source workflow management system “designed to support the automation of complex, service-based and data-intensive processes.” [32]. System was used with success in various fields of science including: bioinformatics, astronomy, medicine, geoinformatics, chemistry, image analysis and even music. Thanks to substantial number of services available to the platform\(^6\), domain independence and ease of use, there is growing interest in Taverna from many parties around the world.

Important part of Taverna is its architecture (Figure 2.10). It is easy to see, that system consists of two separate layers: workflow and engine. The facade pattern were used in the project “to provide external components, notably the monitor and result presentation components (...) with a single point of contact” [32]. This also allows to use engine separately and deploy it in application server, where it can answer calls through REST endpoints\(^7\). More importantly, as it can be seen from Figure 2.10, Taverna provides plugin architecture where its core functionality can be extended using Service Provider Interface. This enables provides to implement specific web service discovery methods and code generation extending standard capabilities. A good example can be Semantic Automated Discovery and Integration (SADI) plugin which allows access from Taverna to semantic services using SADI framework practices [33]. What the plugin does is services discovery and handling inside Taverna. In this way, semantically described web services can be han-

---

\(^6\) Number of web services working with Taverna is in order of tens of thousands and is still growing.

\(^7\) [http://dev.mygrid.org.uk/wiki/display/taverna/Taverna+Server+2.4](http://dev.mygrid.org.uk/wiki/display/taverna/Taverna+Server+2.4)
dled like any other WSDL-described services. Furthermore, the end user is not required to take any other action. Everything is handled by the plugin and later by Taverna engine.

![Overview of Taverna architecture](image)

**Fig. 2.10.** Overview of Taverna architecture [32].

The extensional architecture and the successful application in many different domains were the main reasons for which Taverna workflow management system has been chosen to try orchestrating geospatial web services.
Chapter 3

Theory

3.1. Geospatial service chaining

The need to find easy to use and replicable technique for services chaining and orchestration, has been a source of several papers on the subject. One of the first works about chaining geospatial web services was done by Alameh in 2002 [34]. The paper consisted of three examples of service chaining, with different configurations and user involvement: client-coordinated, static and workflow-managed. Client-coordinated service chaining is, as the name suggests, fully visible to the user who alone is responsible for service execution in a chain. Client serves as an orchestration engine and coordinates everything from service invocation to data handling. This is presented in Figure 3.1. As the Figure shows, user starts from finding needed services (re-projection service, WMS, address matching service) from cataloging service. Next, address matching service is invoked, which finds geographical coordinations of user’s requested address. Coordinates are sent to WMS and image is returned. Finally, re-projection service is invoked to transform a raster image from one spatial reference system to another.

![Diagram of Client-coordinated service chaining](image)

Fig. 3.1. Client-coordinated service chaining. [34]. Numbers represent the order in which given interactions take place.
Another reviewed method presents static chaining with aggregate services. An aggregate service handles the coordination of execution process in a static (pre-defined) way. In this scenario, user only needs to invoke the aggregation service. This lifts the responsibility of metadata and error handling from user and simplifies whole process. On the other hand, by having single point of access, client loses flexibility and control over execution process. Only one scenario programmed into aggregation service is possible. It is, for instance, not possible for the user to change re-projection parameters since he may not even know that re-projection process is taking place. Because of this, aggregate services can be misleading to some users. This is why OGC marks such services with a special flag in its capabilities files [34]. Figure 3.2 shows described scenario.

Final example presented by Alameh is workflow-managed chaining. It is designed to introduce a flexibility of client-coordinated chaining and scalability of static service chaining. At the core of workflow-managed chaining is a mediating service, which can be seen as a “gateway to other services by coordinating between multiple services without necessarily storing any data of their own” [34]. In this scenario, user can create and invoke a chain on his own or use mediating service for that purpose. As shown in Figure 3.3, workflow (mediating) service can be responsible for all steps in a chain from service discovery to data assembly into response. Moreover, mediating service may provide basic functionality such as data manipulation tools or error handling mechanisms, which can be included into chain. As Alameh points out, “mediating services can be considered as specialized versions of existing workflow and process management tools” [34]. Mentioned specialization comes from more complex set of interfaces in OGC applications.

Paper concludes that workflow-managed chaining is the preferred way to execute chains of OGC web services due to high scalability and flexibility of the method. This is also the preferred method in the papers that have followed, which try to find best workflow-

![Fig. 3.2. Static service chaining [34]. Numbers represent the order in which given interactions take place.](image-url)
managed solutions and apply them to geospatial web services. In the absence of solutions that would answer the need for easy and fast OGC web services chaining, several propositions were made.

3.2. Transactional Web Processing Service

In 2008 Schaeffer, basing on Alameh research, pointed out the relative lack of research on workflow modelling despite the importance to geoinformation community [35]. As an answer he has proposed Transactional Web Processing Service (WPS-T), which aimed to be a generic extension to OGC Web Processing Service. Furthermore, working proof-of-concept application was created, which orchestrated mentioned services with BPEL engine. Application was able to provide information on road sections in highly polluted areas. Transactional Web Processing Service extends WPS by two methods: `DeployProcess` and `UndeployProcess` and inherits `GetCapabilities`, `DescribeProcess` and `Execute` methods. It’s interface is described in Table 3.1. WPS was used because, as it has been described in Section 2.2.1, it can provide any type of GIS functionality. Extended interface added transactional capabilities that allowed to dynamically deploy and undeploy processes. Additionally, BPEL WPS-T profile was introduced which, if supported by WPS, would allow to expose BPEL workflows as a simple processes during runtime. In that way WPS-T could be “responsible for either orchestrating or choreographing the workflow or for delegating this task to a third party like a BPEL engine” [35].

3.3. Sensor Web Enablement

Different proposal to chain geo-processing workflows was made by Chen et. al. [36]. His paper presented framework that integrated Sensor Web Enablement services and was a part of NASA Sensor Web Project. The aim of the framework was to retrieve high-resolution imagery and reduce bottleneck for remote-sensing observations. Instead of hiding services
Tab. 3.1. Extended or overridden operations in Transactional Web Processing Service.

<table>
<thead>
<tr>
<th>Operation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DeployProcess</td>
<td>operation allows client to deploy a process according to deployment files. After successful deployment, process is accessible through GetCapabilities operation.</td>
</tr>
<tr>
<td>UndeployProcess</td>
<td>operation allows to undeploy all processes deployed by DeployProcess operation.</td>
</tr>
<tr>
<td>GetCapabilities</td>
<td>overridden operation that additionally contains list of schemas supported by WPS.</td>
</tr>
</tbody>
</table>

behind WPS interface, each service has been described by WSDL document. These were SOS, SPS, and WPS services. Adding WSDL definition allowed to use other tools to create an abstract model with those services that could be later translated into concrete BPEL process and executed.

3.4. PyWPS

The most recent paper from all that have been mentioned about geospatial web services orchestration was written by de Jesus et. al. This work concluded, that usually methods to dynamically manage workflows of geospatial services can be put into one of two general categories [31]:

- WPS service that takes for an input a workflow description and orchestrates services using internal BPEL Engine.

- Adding WSDL description to OGC web service and using them as standard services with BPEL. Orchestration is done in done through BPEL Engine.

Apart from other strengths and weaknesses, none of the methods is suitable for non-technical user [31]. On the contrary, they both are usually used in fields like computer science to prove the concept of OGC web service orchestration. De Jesus et. al. proposed method that solves the issue using different approach, similar to the second strategy on the list above. Using PyWPS framework they extended WPS interface adding WSDL and SOAP extensions and making it available to Taverna workflow manager tool. PyWPS framework is WPS implementation written in Python that provides a way for developers to deploy their geospatial algorithms. Python implementation takes care of parsing input and output. It also describes service with WSDL adding SOAP endpoint. PyWPS can facilitate GRASS GIS functionality which can leave out the implementation part. As the
paper points out, this lowers the required level of computer literacy and helps to focus mainly on model building and data manipulation.

De Jesus et. al. concludes that:

“The Taverna/SCUFL approach to orchestration, where data flow through web services as they become available, is a viable alternative to BPEL (which is more process oriented), and is more flexible with changing workflow structures. SCUFL and its new 2.0 version should be considered as a viable direction in future OGC service orchestration research.” [31]

One of the example of OGC service utilization in Taverna/SCUFL environment can be provided by Biodiversity Virtual e-Laboratory (BioVel). It is a virtual laboratory project run by the consortium of 15 partners from 9 different countries. It allows a researcher to gather and analyze data coming from many different sources. Those can be cross-disciplinary web services put together into workflows. Researcher is able to created his own workflow or use one from the online library. Everything can be done from a web browser. This cuts down time needed for the research and connects to other scientists who may be interested in the project.

BioVel has already been used in a series of researches, such as “Darwin Core: An Evolving Community-Developed Biodiversity Data Standard” or “Development of the Biome-BGC model for simulation of managed herbaceous ecosystems”. Taverna has been chosen as an engine for remote workflow execution. This is, as the site claims, “because of its comprehensive capabilities and its widespread adoption and take-up within scientific communities that increasingly recognise the need to carry out complex computerised analysis and experimental tasks” [37]. Since of its formation, there have been several workflows added to BioVel. At the time of writing only few of them used geospatial data to visualize results on the map. One of the problems was lack of compatible geospatial services that could provide data. For instance, in BioSTIF project, to add another map layer meant to extend the only one compatible WMS service to provide other layers. This is not easy and not scalable solution. Furthermore, because of this, researchers are unable to use other geospatial services and are forced to use the one, they can control.

This is the direction which this thesis explores. It is trying to take step further, and provide native handling of geospatial web services using expendable Taverna architecture. This way it would be possible to use geospatial web services like any other services without the necessity to use PyWPS framework or describing it with WSDL document. This is possible because, openness was one of the key part in design of Taverna architecture [32].
Chapter 4

Experiment

This chapter presents methods used to answer second research question: “What are the methods to make geospatial web services chaining / orchestration more dynamic?”. The solution is achieved based on the guidelines and related works presented in previous chapters.

4.1. Problem analysis

One of the more important aspects of service orchestration is architecture of the services and environment for the orchestration itself. As mentioned before, from three possible service chaining options (client-coordinated, static and workflow-managed), only workflow-managed has the simplicity and control flexibility [34] of the two other options. This is the base and starting point for the solution presented in this paper. This has of course an impact on the architecture which is revealed below. To develop a workflow-managed chaining, a workflow management system is needed. As explained in Section 2.4, Taverna is a system that is very well suited for this type of work with extensible architecture as its main advantage.

Next issue on the way is the presentation of OGC services. Taverna allows to use many types of services including services with REST interface. Because some OGC services provide REST endpoints it means that they could be used without any modifications inside Taverna environment. In fact, this is how BioVel workflows use one WMS service. Of course, this approach has several drawbacks. The biggest one is the fact that not every geospatial service provide REST endpoints. For instance, some SOS services allow to execute GetCapabilities method through REST when other methods can be reached only through SOAP endpoints. This is not the only problem with presented approach. Treating OGC service as any other W3C service makes it more difficult to use it. This is because the user is required to take control of every aspect of method invocation including parsing of XML data. There are also many extensions and solutions present in OGC
services which would be more accessible through native use like transactional operations, results handling / binding etc.

Finally, OGC service methods expose many parameters that are not very important for the end user to set, making whole workflow much more cluttered and complicated to handle. This is the problem presented and described by Alameh, only on a workflow level. Here in workflow-managed environment, it is a conflict in a level of service model abstraction. Low abstraction (or model that is a copy of the service) would allow to use a geospatial service in a workflow with all methods and parameters visible to the user. The drawback of this solution is overcomplicated workflow. On the other side there is highly abstracted model which attempts to handle as much operations as possible by itself giving user the possibility to change only most crucial parameters. Advantage of this solution is clear and easy to maintain workflow. The drawback - little options of configuration.

4.2. Problem statement

In the view of the related works presented in Chapter 3, this study tries to ascertain how effective would geoservice orchestration proved to be using Taverna workflow management service. Contrary to other works that have been using Taverna, this one tries to automatize service orchestration providing native support for OGC services. In this way, it would be easier to add and use geospatial web services in workflows, using them as any other services. Furthermore, if proven feasible, native support would allow non-technical audience to compose and execute workflows. This would be an answer to further study advices formulated by Alameh, Schaeffer, Chen and others. Finally, thesis will try to answer the question of the potential of Taverna compared to BPEL solutions.

4.3. Experiment design

Those are the most challenging problems that needed to be addressed. With Taverna as a workflow management system, those issues can be addressed and the compromise between complexity and ease of use can be reached. This can be done using Taverna extensibility property. It was mentioned before that Taverna provide means to extend its capabilities via external plugins. Plugins can hide as much complexity behind the scenes as needed shifting the burden of configuration to developer rather than researcher using the product. Moreover, because Taverna workbench has very rich presentation layer, service properties can be displayed more clearly to the user without the need to extract them from XML files. Another advantage addressing the issue of complexity versus configuration are the methods in workbench to further configure any given service. This way, all additional options can be changed, maintaining clear and easy workflow. Worth noting is the fact that hidden complexity has no effect on output data, nor provides less options.
To help visualize discussed concepts, relevant areas were marked on main Taverna window in Figure 4.1. Going from the beginning, service tree area is responsible for service selection. All services can be added to the workflow from the tree of services. The tree can be configured by plugin implementation which allows to choose best layout for given type of service. As a result, related methods can be grouped together in branches to present consistent structure. Each leaf can represent a single service but this is not always a rule. When service is complex enough with many methods, each method can be represented as a leaf. This way the whole structure is more granular and may be also more clear. It is then possible to compose workflow from different methods treating them as building blocks.

After choosing appropriate service, it is added to the workflow by the user. Selecting service in the workflow reveals its properties in the service details panel (Fig. 4.1). User can review service, read its constrains, properties etc. Each service is presented as a building block with input and output ports (Figure 4.2). These ports can handle text as well as binary type data. Furthermore, their number and goal can be changed by developer and end user. Developer programs logic of ports and their default layout. He can also hide some functionality under configuration button (Fig. 4.1) making, for instance, less needed I/O ports visible only on user request. This of course makes developer responsible for handling default values from hidden ports.

Last but not least, there is also an issue in adding publicly available OGC service
4.4. Requirements analysis

In order to satisfy proof-of-concept implementation objective, following requirements have been selected:

- Taverna support for OGC SOS and WMS services should be provided in form of plugins so the user is not aware of the implementation details.

- User should be able to install plugins from remote repository.

- When a new version of plugin comes out, user should be notified and be able to update plugin.

- After installation user should be able to add OpenGIS SOS / WMS services by providing full URL to the service.

- Adding new service, should invoke \texttt{GetCapabilities} method on the service and build a tree based on it.

- Service tree in case of SOS should contains observation properties sorted by observation offerings.

- Service tree in case of WMS should contains layers.

In conclusion there are several levels on which OGC services orchestration can be streamlined. All of them are examined and discussed in this chapter.
• Selecting WMS / SOS element on the workflow should present this object’s details retrieved from GetCapabilities document.

• There should be implemented basic validation of workflow and service state.

• SOS observation property element when invoked should return XML answer from GetObservation method.

• WMS layer element when invoked should return text URL with invocation and PNG image from GetMap method.

• It should be possible to create a workflow containing both SOS and WMS services.

Above requirements were written to fulfill aims and objectives presented in Section 1.1. They are based on restrictions imposed by Taverna workbench and used methods formulated before.

4.5. Architecture design

This section presents architecture of proposed solution. In order to present any orchestration between OGC services there are at least two different services needed. For this reason, there have been chosen two types of services: Sensor Observation Service and Web Map Service. The reason behind choosing SOS service was its seemingly simple architecture. On the other hand WMS services are in high demand from the geospatial community. For this reason, it is often used as an example of geospatial service. Usage of those two types of services made it easier to compare it with solutions presented in other papers.

Using Taverna meant, that most of needed components were provided. To add support for WMS and SOS services, Taverna had to be extended through one of the extension points. In Taverna this can be done using plugins. Each plugin supports one type of service. There are already many plugins that provide support for services like: SOAP/WSDL, REST and many others. Installation of plugin requires remote Maven repository. After installation, plugin operates on two levels: engine and workbench (see Figure 2.10). Because of this, in reality one plugin consists of two separate parts: invocation and service discovery plugin. Service invocation plugin can operate alone whereas discovery plugin has invocation plugin as its dependency.

Service invocation plugin operates on engine level. Generally, if there is a need to add support for another type of service it is done by adding another service invocation implementation. Because, as mentioned before, plugin works on engine level it does not require any additional dependencies from graphical user interface. This is an important fact when user would like to extract an engine with additional service support and put it in application server. Functional requirements from Service invocation plugin are:
• Define processor’s input / output ports - each processor define it’s ports to communicate with other processors. Ports can be static or dynamic.

• Service invocation - connecting with the service upon workflow execution. Anytime processor is being run it is responsible to connect to the service, and get results.

• Results handling - parsing and checking results received from service.

With requirements in mind, activity plugin has the structure of packages presented in Figure 4.3. Starting from the top, `pl.wroc.pwr.activities.sos` package contains core classes responsible for requirements described above. There are three additional packages: `exception`, `util` and `model`. At the moment exception package contains only one exception class designed to indicate that process output type is not yet supported. Util package contains utility classes aimed mainly at REST/SOAP requests building. Such requests are build and executed every time a processor is scheduled to run. More about methods to create a request and run it will be covered in section below. Finally model package contains classes designed to store service invocation results, invocation requests etc. Section 4.5.1 describes some of the reasoning behind structure and state of the model.

On the other side there is service discovery plugin working on GUI layer. This plugin is required to add service to the workflow. It is not required however to execute workflow itself. Below are the functional requirements from plugin:

• Service discovery.

• Processor configuration - plugin can allow to further tune processor with additional options

• Processor state validation - validate if all requirements are fulfilled (i.e. required input ports are connected or inner state of processor is valid).

Because service invocation plugin is a dependency to a discovery plugin, all packages (and contained classes) are available for classes in discovery plugin. This dependency is one-way only. Dependency helps to initialize the model, as well as send execution requests from GUI. The overall structure is simple and does not contain any additional layers of communication. Figure 4.4 presents package diagram of service discovery plugin.

![Fig. 4.3. SOS service invocation plugin package diagram.](image)
Starting from root package `pl.wroc.pwr.activities.sos.ui` there are no classes contained in this package. Although the package name can suggest that it is a part of engine, this is not the case in here because this package belongs to different project. Next, there is configuration package. It contains classes responsible for user interface and logic behind Configuration window. This is the case for SOS plugin where user can set output response format of GetObservation method (Figure 4.5). Another package is serviceprovider. It contains classes that represent service state. Some classes take part in service initialization in the moment, that it is first added to Taverna workbench via “Add service” button (Figure 4.1). Initialization is done by checking given URL and trying to execute GetCapabilities method on a service. This is done regardless of the type OGC service interface because they all have must have this method implemented. If operation succeeds then service tree is updated and service is available to receive method calls.

Method calls requests are coming from GUI. Programmable part of user interface is stored in view package. The most visible element from this package is processor description (Figure 4.6). Implementation of this part is covered in Section 4.5.1.

Last package that needs to be introduced is handlers package. This package contains XML handlers capable of parsing documents they were made for. In case of SOS service plugin there is only one handler to parse GetCapabilities answer. Handlers are used with Simple API for XML (SAX) to parse XML data using event-based API. This will be covered in more detail later on.

In order to find plugin that can handle given process on the workflow Taverna uses Service Provider Interface (SPI). It is a Java technique where a service is defined as “a well-known set of interfaces and (usually abstract) classes”[38]. This solution provides
scalability to plugin mechanism, because adding another plugin means to provide another implementation of service provider interface. During startup, all services are discovered and instantiated. Later, during i.e. workflow execution, Taverna asks services through SPI if they support given type of processor. If service does support a processor, then the execution is delegated to that service.

Another important point of communication is located between plugin and geospatial web service. The lack of any specific interface is here made on purpose because all OGC web services are based on OWS interface. This means, that every OGC service is required to implement `GetCapabilities` method (as mentioned above in Chapter 3). This is the default method of communication with OGC service and a starting point for configuration. `GetCapabilities` response contains all required information for successful communication with the service including version, method names, allowed method parameters, service metadata and other. Despite the fact that every service is required to implement `GetCapabilities` method, returned object usually differ from service to service (response can be based on different type of schema or different version of the same schema). This is troublesome to say the least, and requires developer to implement handlers for all versions of schema if he want to create an universal solution. In case of developed proof-of-concept this problem was addressed by leaving space for scalability. If, for instance, someone would like to add support for second version of SOS standard, s/he would need to create specific model for a given version and make use of available extensions.

### 4.5.1. Implementation and Test

To ensure that written code is working as expected, proper configuration had to prepared. First of all, Java language was selected, because whole Taverna environment is running
4.5. ARCHITECTURE DESIGN

More specific, compliance with Java in version 6 had to be maintained not to force end-users to make any changes in their installations. As a software control system was chosen Git - free and open source distributed version control system. It was chosen because of its popularity, ease of use, rich toolset, and many compatible applications. One of such application was GitHub - web-based hosting service which allowed to create a number of remote repositories storing committed code [39].

GitHub itself is a very well designed service allowing user to connect it with a number of other web-based applications. Proof-of-concept made use of this feature by using Travis-CI. This is a “hosted continuous integration service for the open source community” [40]. Every commit to GitHub repository, starts building of updated code. If code is not compiling or some of the tests have failed, Travis will fail the build and notify the developer. Travis is configured with a single file called “travis.yml” present in repository. Its content is presented in Listing 4.1. File indicates to Travis that it should use Maven with goals clean and install. It should also repeat whole building process for oracledjdk7 and openjdk6. This is very handy option because it helps to provide support across a wide range of different user configurations.

```
language: java
script: mvn clean install
jdk:
  - oracledjdk7
  - openjdk6
```

Listing 4.1. “travis.yml” file for Travis configuration.

With Github as a remote Git repository and Travis as a version control system, whole development and integration process was moved to the cloud. This has been beneficial for development process because there was no need to set up another server only to host control version manager and continuous-integration system. What is more, code that have been written was replicated in several places so there was no fear of loosing it by accidental disk failure.

Apart from mentioned tools, the rest of the project has a standard configuration. Project was developed using Eclipse IDE and was build with Maven. Because plugins are normally installed in Taverna from Maven repository, Maven POM files had to be configured to allow deploy execution step to send all files into remote server. Temporarily, as a remote server was chosen Dropbox service, another cloud solution aimed mainly at file storage. This temporal solution has proven to be helpful. Because part of public folders on Dropbox does not change it’s address dynamically, it can be used reliably and without any problems as a Maven repository. After certain changes in Maven POM files, deployment could be performed and files started to be uploaded to Dropbox folder. General layout of Maven POM file with indicated repository settings is in Listing 4.2.
With remote Maven repository it was possible to install and update service plugins. The complete path to accomplish that task is presented below. First step is to add an address with Maven repository and the deployed plugins (Figure 4.9). After this it is possible to install a plugin (Figure 4.7). If there is more than one plugin deployed in a single repository, Taverna will show both as available to install. During installation process all jar files are being downloaded from Maven repositories. After installation and Taverna restart it is possible to use plugin. If during that time, POM version will change and new jar will be deployed to the repository, Taverna will inform its user that an update is ready to be installed (Figure 4.8).
4.6. Evaluation and related works

As a result of described before proof-of-concept, two plugins have been created. It was discussed that those plugins were supposed to enable OGC services chaining / orchestration in Taverna environment. Therefore, after finishing development work, end-user work started. As an end-user a researcher would like to try and execute OGC services in real environment connecting them with other services. This scenario was followed and workflow was build containing one SOS. Figure 4.10 presents a workflow build with SOS service returning *waterlevel* values using *GetObservation* method from SOS interface.
4.6. EVALUATION AND RELATED WORKS

CHAPTER 4. EXPERIMENT

Fig. 4.9. Adding plugin site to update site list.

Fig. 4.10. Workflow returning waterlevel from SOS plugins.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<om:ObservationCollection gml:id="oc\_1360735276149" xsi:schemaLocation="http://www.opengis.net/om/1.0\,http://schemas.opengis.net/om/1.0.0/extensions/observationSpecialization_constraint.xsd\,http://www.opengis.net/sampling/1.0\,http://schemas.opengis.net/sampling/1.0.0/sampling.xsd"
xmlns:om="http://www.opengis.net/om/1.0" xmlns:gml="http://www.opengis.net/gml"
xmlns:swe="http://www.opengis.net/swe/1.0.1">
  <gml:boundedBy>
    <gml:Envelope srsName="urn:ogc:def:crs:EPSG\::4326">
      <gml:lowerCorner>52.9 7.52</gml:lowerCorner>
      <gml:upperCorner>52.9 7.52</gml:upperCorner>
    </gml:Envelope>
  </gml:boundedBy>
  <om:member>
    <om:Observation gml:id="go\_1360735276149">
      <!-- Additional XML content here -->
    </om:Observation>
  </om:member>
</om:ObservationCollection>
```
In order to test WMS plugin in practice, another workflow was created containing WMS processor representing GetMap method (Figure 4.11). Workflow defined two output ports: invocation and map. Invocation port returns URL address used to as an input to WMS service. This is just for debugging properties and allows developer to check manually communication between plugin and WMS service. On the other hand, map port returns binary data containing map image. After workflow execution map port returned picture shown in Figure 4.12.

Finally to test more realistic use case, a workflow with both SOS and WMS services was created. Complete workflow diagram extracted from Taverna workbench is in Figure 4.13. The main purpose of this workflow was to extract position of SOS sensor and values registered in requested user time period. The version of SOS interface for this particular service was 1.0.0. Service offered two ObservationOffering properties:

---

1For this particular workflow a 52°North SOS service was used available at the time of writing under following address: [http://sensorweb.demo.52north.org/52nSOSv3.2.1/sos](http://sensorweb.demo.52north.org/52nSOSv3.2.1/sos)
4.6. EVALUATION AND RELATED WORKS

Fig. 4.12. Map result from WMS processor.

- **GAUGE_HEIGHT** - the water level in a river.
- **WATER_SPEED** - the water speed at a gage in a river.

In this workflow **GAUGE_HEIGHT** property was used with physical location of the sensor. It is visible at the top of the workflow as white rectangle with name **waterlevel**. There are two input values that provide start and stop time, marking the duration for value retrieval. As an output there were also two values: water level and position of the sensor. Those two values were together placed inside XML file. In order to parse and extract each value, there were created two nested workflows with self-explanatory names: **Extract_values** and **Extract_position**. Those workflows used mainly two types of services: XML XPath service and String split service. Those two operation helped in creating table with SOS values and extract position of service. Nested workflows gives additionally the possibility to test a given part beforehand. This way user can be sure, that certain part must be functional and in case of errors, s/he should search elsewhere. Nested workflow can also be used in another workflow so there is also modularity gain here. Extracted SOS water height measurements were passed down to Workflow output port **observedValues**. On the other hand, position was used as an area input for WMS plugin \(^2\) which returned map containing SOS position, as well as invocation used to obtain this map. Because map would show only a single point, user value was introduced as an input to construct bounding square containing SOS service position in the middle. Every time user provides a number, it is being used by to widen the selected area. As expected, workflow finished successfully and returned formated measurement values with map area of SOS sensor, widened by value provided by user.

As can be seen from the workflows, OGC plugins hide service invocation logic, providing easy way to mix OGC services with other services without further adjustments. From user perspective, a SOS or WMS service is no different from any other service in the

---

\(^2\)WMS service from National Oceanic and Atmospheric Administration was used in this workflow (available at the time of writing under this address: [http://nowcoast.noaa.gov/wms/com.esri.wms.Esrimap/obs](http://nowcoast.noaa.gov/wms/com.esri.wms.Esrimap/obs))
service tree. This is different from previous attempts in a number of ways. If compared with latest ones like BioVel, user did not need to read \texttt{GetCapabilities} file prior setting parameters to REST interface. Examination of processor details was all that was required to properly configure processor. What is more, OGC services used in this example were simply found on the internet. There was no additional configuration required, no need to own a service, to annotate it with WSDL or hide under WPS interface. As long as version of service is supported it can be used in a workflow.
Chapter 5

Discussion

Despite the advantages of presented solution and the working proof-of-concept, not every aspect was perfect. This statement applies equally for design as well as the implementation part. Considering that, this chapter aims to discuss the problems that were encountered during the development process as well as downsides of the proposed solution.

First of all, the most fundamental question must be asked about the OGC standards themselves: what are the advantages of using standards? The one obvious answer is that standards can guarantee interoperability between different systems using common contract defined by the standard. On the other hand, because of fast changing requirements and technology, standards need to keep up with the current demand. If standard is not updated or is updated too slowly, it is abandoned by the interested parties and alternative solutions are used. Then an interesting process starts: the standard is trying to catch up, and includes best practices already in use. This scenario is especially common in IT world with new technologies emerging every month. This might be also a reason why OGC standards are not used by all companies and why these companies rely on their own solutions. Open Street Map (OSM) is a good example of such practice. They argument the lack of OGC compliance with the following statement:

“Unlike the business world, OSM does not view standards compliance as a goal in itself; i.e. if we are to use a standard, it must have clear an direct benefits for the project - otherwise nobody will code it or assign resources to it. While this does not apply to each and every OGC standard, on the whole they range from “cumbersome” to “useless” for OSM and thus are disregarded.”

One point made in the statement is that some OGC standards are “cumbersome”. It is tempting to use this word based on experience gained from implementation of proof-of-concept for several reasons. One of them can be the lack of compliance between different

1The whole discussion can be found under this address:
http://help.openstreetmap.org/questions/981/ogc-and-interoperability/990
versions of the same standard. It is therefore impossible to use SOS in version 1.0.0 by the software compatible with version 2.0.0. During designing proof-of-concept there was a decision to implement the most recent versions of standards. This was changed after noticing that there are hardly any public SOS services using 2.0.0 version of the interface. Since the version are not compatible it would be impossible to test the solution. There was very similar case with WMS standard. This of course doubles the work required to do.

Second problem encountered, this time during implementation, was complicated schema design of both standards. This may be explained partly by the number of different formats supported by the interfaces. Nevertheless, the simple translation of SOS XML schema to classes using Java Architecture for XML Binding (JAXB) resulted in 719 different classes. It was then impossible to bind XML to an object during runtime because of not enough memory when creating Data Object Model.

Finally, because the standards allow the choice of use for many different technologies, support for them must be provided by developer. As an example the RESTful communication with OGC services is optional and the information which method supports SOAP, which supports both SOAP and REST is provided in metadata in GetCapabilities document. To be compatible with most of the services it is best to implement both methods of communication.

What is more, not all of the methods mentioned in the standard needs to be implemented in the actual service. For instance, WMS service is required to implement GetCapabilities and GetMap. All other methods are optional. Situation is similar in other interfaces. This must have been all accounted for during implementation of proof-of-concept.

Adding to that, there are alternative ways of geographical content distribution to OGC standards. Apart from mentioned OpenStreetMap, there is for instance commercial software ArcGIS developed by Ersi. ArcGIS apart from map distribution provides a ways for man development, presentation and analysis. ArcGIS supports REST and SOAP endpoints to its server solutions. There is additional option to provide an WMS endpoint but it is optional.
Chapter 6

Summary

The aim of this paper was to address the automation in geospatial data processing. In order to do it, three research questions were formulated (Section 1.2). All of the questions were answered and the final result in form of a working proof-of-concept was achieved.

The first research question asked “What have been already done to automate data processing in geospatial web services?” To find an answer an extensive literature study was conducted and based on information gathered the content of Chapter 2 was written. It includes the description of the current state of technology used in two separate environments: W3C / OASIS and OGC. For each environment, a technology stack is presented. Additionally, process of service chaining and orchestration was explained. This chapter also provides remarks on why service orchestration for geospatial services cannot be executed in the same way as for W3C / OASIS services. An answer to the research question is described in Section 2.3 (Workflows). This section mentions technologies directly connected with automation of data processing in geospatial web services. Those are: workflows, languages to define them and taverna workflow management system. The answer to first research question shows research gap, needed to be explored in two following questions.

Second research question tried to decide “What are the methods to make geospatial web services chaining / orchestration more dynamic?”. A literature review was performed and several papers have been selected. Articles contained not only the most recent development, but also the initial problem analysis. As a result Chapter 3 (Theory) was written. It started with a reference to the Alameh’s work, who divided service chaining into three different types: client-coordinated, static and workflow-managed [34]. Workflow-managed chaining was chosen as an optimal way giving both ease of use and flexibility to the end user. Next, more recent papers presenting the methods of workflow-managed chaining usage with OGC services were analyzed. The answer to the second research question opened direction for further studies. They resulted, among the other, with identification of Taverna as a very capable workflow management system that has a great potential for web
services orchestration. Based on current papers, a new direction was chosen. Having this stated and based on published papers, a final decision was made on which direction to take in order to realize a practical part of this work. The target became an extension of Taverna workflow management system by means of plug-in implementation that would support the use of OGC services within it as any other services.

This direction was further explored to answer the final research question: “How effective is the new approach proposed in this thesis?” The main part of the work done in this scope was an implementation of a working proof-of-concept. It took a form of the workflow presented in Figure 4.13. The workflow can be build and run thanks to the implementation of two plugins offering support for WMS and SOS service interfaces. These plugins were designed and developed by the author of this thesis. Finally, the workflow was executed and the goal of the thesis could be achieved. OGC services worked together with other types of services, and there was no difference between them from the user point of view.

It has been shown that orchestration of OGC services is not only possible but also quite easy, given that proper tools are created beforehand. Because Taverna provides a lot of flexibility and can be extended in many ways, this proves not to be a problem, as was presented through the paper. Extending Taverna to support OGC services enable scientists to build even more complex workflows and solutions. Adding support for additional OGC services is a direction for future research.
Bibliography


[38] Oracle. JAR File Specification. URL http://docs.oracle.com/javase/1.5.0/docs/guide/jar/jar.html.
