WSDL Workshop

Semantic web application in HTML5 for the discovery, construction and analysis of workflows

Rafael Adolfo Nozal Cañadas

20.11.2013

Master thesis
Department of Informatics
University of Bergen

1 Thanks

I want to acknowledge the aid that the following people gave me during the making of my master:

Steinar Heldal; who infinite patient overcome the infinite paperwork of Spanish bureaucracy during my enlisting process, and who also aid me into getting some work as teacher assistance during this time.

Dr. Ingvar Eidhammer: First contact I had with bioinformatic and who explained the essentials about this world during my first year. Also aiding and guiding me in the making of this thesis.

Dr. Pål Puntervoll: I don't have enough papers to begin to make justice about how much his guidance and support helps during this time. I'm hope to work with him in the future again and maybe have some biology courses under his supervision.

Dr. Natalie Reuters: For the course she tough about applied bioinformatics. Help me to understand and have a higher perspective on how everything works together. Also to open my head on how to work in a biology environment. Being my background of pure informatic engineering, seeing thing from a non mathematical perspective could be uncanny and scary the first time; unless you have her as a teacher.

Matúš Kalaš: Helped me with the understanding of WSDL in general and let me use his EDAM work which is one of the main pillars on which this thesis is based.

2 Table of Contents

Contents

3 List of figures

List of Figures

4 Abstract

WSDL-Workshop is a HTML5 web application for the discovery and exploration of web services and for analyzing the compatibility between web services. This is the result of a mathematical model developed from WSDL1.1. The program provides a graphical user interface and let the user build a workflow composed of services described with WSDL1.1 and tells if:

- An output is compatible with an input.
- It is correct to link an output with an input.
- It is correct to link a given operation after another.
- If many operations correctly linked together still make sense as a group.

In order to do that the WSDLs must have semantic annotations so the computer can recognize what is the purpose of certain data or operation. WSDL-Workshop uses the EDAM Ontology as a reference for semantic concepts.

In the discovery aspect; for a given set of WSDLs you can find services by filtering by operation name, input or output names, or semantic annotations. For a given operation output it can also filter by WSDL which have inputs which are correct to link with that output. For a given operation it can also filter by operations which are correct to link after.

5 Introduction

In computing there is a concept call a "black-box". When you implement some functionality you specify the inputs of your function, the output of your function and what the function does; but you don't tell how does it.

Similar to this concept, in the web, we can find what is known as "web services". These are programs where we know what they do, what we are suppose to give to them, and what will be received. Yet we don't known how they do it.

We can make a chain of such programs, and give the output of program A to the input of program B. It is the responsibility of the designer that the output of program A is compatible with the input of program B. The designer must also have in mind some kind of objective of why is he doing that, and what kind of meaning the output of B will have. For example, we can have program A that subtract two numbers and square the result, and program B which add two numbers and square-root the result. In principle they don't seem to have anything in common but if we link program A with program B, we end up with the euclidean distance between two points given as an input to the programs A.

In another example and using such web services, we can ask a weather forecast service for the temperature of a city. We can use the number we receive as a z-score to calculate the probability of having that temperature. If you know a little of statistic (or meteorology), you will realize that this is a huge mistake. The correct way would be to, given a normal distribution, with an average and standard deviation; normalize the temperature received, and that will be your z-score. Yet is possible to do it the wrong way and get a z-score of $+25^{\circ}$C from a spring day in Nassau (The Bahamas), which will give you an area of 0.99999999... and the result will be that it is impossible to get a temperature over that temperature ever. In reality, a $+25^{\circ}$C day in The Bahamas is a very normal day; so the designer makes the mistake of given the incorrect concept to another function, even thought the input is perfectly compatible and could be any real number.

For a computer is it trivial to understand if an output is compatible with an input. If you try to give a string to an integer the computer will complain and won't even let you do the operation. But is it possible for a computer to realize whether compatible data make sense together?

We can do it if the data have their meaning as extra information. This is known as a concept which has semantics associated with it. We can have different concepts such as inputs and outputs, operations, datatypes, and so on. At a data level we can tell that a car's license plate (string) is not compatible with a street address (also string). At the operation level we can tell that an operation that takes a flight reference and gives you back destination city and time, makes sense to link with another operation that books hotels in a given city and time.

Furthermore, we can try to make sense of a chain of operation as a whole. Having the booking reference we can find a hotel in a city. Having a hotel we can search for critics. Having a critic we can search the text for data mining

concepts. Having concepts we can find their etymology. Having etymologies we can find for near historical events. And we can continue to infinity.

All of these concepts can be imported into a biological context, where the operations are not hotel booking but searching relevant places in a DNA, neither look for a parking spot but find similar proteins. In that case you will have a semantic web application centered on biological concepts, which is what this thesis is all about.

6 Background

In biology you frequently make use of PC programs that essentially analyze some data and give you some new data in return. For example you have a protein, which is a datatype of list of residues (encoded as a character each). Then you can perform an operation called BLAST that will get you similar proteins within the database. And in return you get new data with information about that protein, as for example species which have the same or similar protein and how far away are they in the evolution tree with respect the initial protein. Or you could have a 3D structure defined, which is a representation of, for example, a physical enzyme. It could be interesting to find if some external atoms fit somewhere inside the structure, so you use a program to search for cavities; now you have some coordinates with points and surfaces where an ion of Calcium could fit inside.

This collection of programs for data retrieving, data storage, analysis, discovery, exploration and ultimately gaining a bit more of knowledge by using them, is what is known a bioinformatics. Here you will find two main groups of people; those who develop software to help others and those who use that software in research. This thesis belong to the first group, and I'm developing a web application to help the second group to analyze which combination of programs should be used to meet their objectives.

Lets take a look to some other examples of programs which I will later refer to:

The Jaspar Database [1] contains a collection of DNA binding sites [2]. In a DNA chain there are special areas where other molecules can attach to it. That concept has significant importance because you can develop compounds that prevent or enhance the binding of the DNA with that molecule. That could make it possible to encourage or repress such bindings, ultimately changing the health of a multicellular organism such a human being.

ClustalW2 Phylogeny [3] is a service that allow you to create phylogenetic trees. This is a type of representation that gives us a visual overview of how far away are sequences from each other. One well known example is the tree of life which gives a summary of the evolution of the species. Is not only for that; a phylogenetic tree also tells us differences of group of sequences and allows for different types of metrics to calculate similarities. This is important because if we find a protein that is similar to another, it is very possible that the second protein have the same function as the first one. So we can develop a substitute that have the same active principle but maybe with less side effects, or others that are less dangerous for a patient. Or for example could be that a given protein which is produced in a very rare endanger or extinct specie, has a sister protein extremely common in nature, which will drop the price of a medicine.

In these examples we see that we need to find something that fits in a given binding site, or that we need to retrieve the species of a sequence given by the phylogenetic tree. You often use the data gathered from a program as an input to another program. And so the chain continue from your original idea until you have the final desired data acquired by the number of transformations you

considered necessary. This chain of programs and transformations is known as workflow.

Making a workflow is not a trivial task. There are several alternatives out there to find and construct such chain [4] [5]. And there has been an extensive work into discovering relevant programs to be use in such constructions [6]; not only in bioinformatics but in different areas as well.

But discovering the programs is not good enough. You also need to find possible relationships between them. Is program A compatible with program B or do I have to make manual tweaks to make it work? Is program A related to DNA analysis or am I actually using a program that have nothing to do with it? Overall; is my chain of programs good for my interest?

There are programs out there known as "Workbenches". These programs allow you to import a web service, send some data as input and receive the output. Instead of receiving the output you can make the program give the output to another web service. And so you can make a workflow with many services and receive only the last output, which it is what is relevant to you. Once again this is a challenge in the bioinformatic area and there are several alternatives out there.

Taverna [7] [8] for example provide you with a graphical user interface and allow you to build a workflow and execute it. Taverna uses a decentralized approach and sends and receives data from services around the world. Later on you can share your results in a social network call myExperiment [9] [10], and other fellow scientist can copy your finding or point to errors in the methodology. A different approach of this is Galaxy[11] [12]. Galaxy work similar to Taverna, it allow you to make track of services and keep track of where each output was connected. Such a log of techniques is also a workflow. Galaxy however is a monolithic application and only let you use services inside Galaxy. An advantage is that your results are much more likely to be reproduced again at the expenses of having a small varieties of services. Taverna depends on third party services which can change their implementation or just disappear from the Internet, but you have richest pool of resources. A mixture of the two could be eSysbio,[13] [14] [15] which keep track of the executions to aid reproducibility as in Galaxy, plus allow you to add new services and R scripts, and add sharing options like in Taverna and myExperiment. There are more workbenches such as Chipster [16] [17] or GenePattern2.0 [18] [19]. Even the European Union is pushing intro create a common research environment such as Elixir [20] which will also have its own workbench similar to Galaxy.

While all of these alternatives are great, none of them provide a semantic analysis of their services, so none of them allow you to search for services that are relevant for you. You are responsible to look out there for services, and you are responsible to make manually workflows that works and make sense.

If we want the computer to work for us and be able to do it automatically we first need to create a set of knowledge, a web of relationship between concepts to be able to tell what is relevant for a given topic or objective. This is what an ontology is. An ontology tells you what is the relationship between "car"

and "house". Navigating the concept "car" we could find a relationship called "parking place" which point into "garage", while "house" can have a relationship of "composed of" which point to "garage"; and you can even want to discover the different types of garages that could be. But instead of having an ontology centered on architecture like in this example, we need an ontology centered in bioinformatics concepts. Such an ontology is the EDAM ontology [21].

7 Aims

This thesis aims to develop an experimental system called WSDL-Workshop which functionality is to analyze whether different services are compatible. It will not only check if datatypes match, which is what system like Taverna and Galaxy do; but it also will pay special attention if they are semantically compatible. It will also allow to search relevant services which are compatible with a selected service. Summarizing, it will try to bring understanding of concepts to a computer so it is able to construct good workflows automatically. A task that until now was exclusive of humans with experience in biology. Following you can see the hightlights of each part.

Develop a method to be able to tell if:

- A data is compatible with another data.
- An operation is compatible with another operation.
- For a given chain of compatible operations if there are alternative ways of doing the same task and determine which one is best.
- For a given task find automatically the best chain of operations (workflow).

Create an application to test that method. The application must have the following requirements.

- Easy access for the user.
- Can be easily copied and modify by others.
- Help the user in the construction of a workflow.
- Help the user discover web services suitable for that workflow.

In this thesis you will find a list of technologies and concepts used in here and several description to understand each of them, the theoretical model developed using these technologies, the program developed in order to test this theoretical model, a discussion about choices and problems encounters during the making, and the conclusion. The end product is a very promising application that allows for automatic workflows construction.

8 Technologies

Here we will talk about the different concepts and previous knowledge which you need to comprehend in order to understand the theoretical part. Also to understand some decisions made during the design of this project. You will find explanations about HTML, XML, XSD, WSDL, EDAM ontology, SAWSDL, CSS3, JavaScript, HTML5 and the Acid Test and SOAP protocol.

8.1 HTML

HTML is a language used to describe information and the way that it is displayed [22]. Generally the main function of this language is to describe a web page in a structural manner, so later on the web browser can read it, interpret it, and display it on a screen for a human.

The structure of the language consist in "tags" which usually come in pairs known as open and close tags. Whatever is in between these tags is affected by the kind of tag that envelop it. For example <ul> will list the text which is inside with no particular order; each item inside the list must be tagged with . The tag format the text so is show in bold. So the following code: <ul> Something in HTML will look like something similar to this in your web browser:

- Something
- **In**
- **HTML**

I must emphasize the word 'similar' because the language is just a guideline for your web browser. This will display a list with the last two elements in bold; but it doesn't says how big should be the markers, or the size of the text, or for example we haven't designated the default style of font for the text. So all of those minor options will be up for your web browser to decide.

Beside that, there are other aspects that a web browser must comply to be considered a standard and fully compatible with HTML language. We will discuss about it in section 8.10 in page 29

8.2 XML

XML is another language very similar to HTML that describe information, but not how is displayed [23].

The key elements of XML is that also features a tag system and can be read easily by humans. For example if you want to encode the information about books we have in our house we could represent the data like this:

<library> <book> <title> My thesis </title> <year> 2013 </year> </book> <book> <title> Somebody else thesis </title> <author> Finn Author </author> <year> 2004 </year> <location> Shell A </location> </book> </library>

So as you can see the information and the meaning of each tag is quite obvious. The information is enclosed nicely between the <book> tag so we know that every information about a book still refers to the same book until we close the tag.

There are more complex structures than this one that could be read in the XML specifications. In this case I'm going to describe how to describe an application using XML which is essentially what a WSDL is.

One component of the XML is the XSD [24], which is also known as 'schema'. This is a set of metadata that describes a part of the XML document. This is usually achieved by an independent XSD file where you can find the specification for the concepts you are talking about. In the context of this thesis, an XSD is use to describe datatypes. For example what you call a 'matrix' could be a simple 3x3 integer matrix described in a XSD called mymatrix.xsd; which is referenced in an XML file. However for me, a matrix could be a 5xNx9 amino acids layout, where even the amino acid datatype is described in another XSD. Ideally these XSDs are unique and when you talk about your matrix you will define the datatype as for example 'm33:matrix', and when I talk about my matrix I will identify it as 'aminoM:matrix', so it is possible to see quite easily that they are both matrices, but have nothing to do one with the other. This is not always the case, so every XSD must refer to a unique URL. In that case, even if we have different schema names as is the case for 'xs:string' and 'xsd:string', as long as xs and xsd refers to `http://www.w3.org/2001/XMLSchema` they are consider the same datatype.

8.3 SOAP

This stands for Simple Object Access Protocol [25]. We want computers to talk to each other, independently of whether we are trying to send the input of a WSDL, receive an email or have an open stream of video; and to do that we need to establish a protocol that both computers can understand. SOAP is a protocol optimized to transmit XML data. It has nothing to do with WSDL by itself, we can encode any data in XML so it can be use to send anything. There are much better alternatives depending of your source of the data; for example it can be use to transmit video but that would be quite inefficient. However, WSDL is developed in XML, and it is very convenient to use SOAP to transmit data described in the WSDL files, XSD files, inputs, outputs, and so on. As a result of this the two of them, WSDL and SOAP, has become closely interconnected.

8.4 WSDL

This is an extension of the XML language. It stand for Web Service Description Language and the objective is to describe an application that is, typically, accessible over the Internet.

The point of this language is to describe how the application works and interacts. It provides definitions to the different datatypes. Which data goes

in and out of the application. How the data is related with respect to each operation. Which operations can be performed with the service. And of course how to access the service itself.

A web service is a set of functionalities that are available on the Internet. For example, booking a plane ticket is a web service that needs a date and some data from your credit card. In return it will give you a booking reference. In a more scientific biology related context, BLAST is also a web service. You just need to fill all the forms (the inputs) and the application will return the results (the output).

There is a lot of ways of defining a web service. The most usual technology is that you have your web service (WS) working in your private server, and you provide the interface to the user. One of many ways of doing that is in a WSDL file. A user with a WSDL file can use your WS since the file will contain all the proper information in order to communicate with the WS, as in which inputs and outputs are expected, or which protocols and addresses are needed.

Currently there are two versions, WSDL 1.1 [26] and WSDL 2.0 [27]. both of them are very similar from an abstract point of view, as in both of them have the exact same features that I described before. The main difference between the two of then is that they don't follow the same syntax, and the structure the document is different; but both have the same abstract concepts of describing inputs/output and how to commit/fetch them. In this thesis I'm using WSDL 1.1.

Please visit the W3 references to find the excruciating documentation on how the WSDL works. Here I will give an overview with an example to make you understand the technology.

8.4.1 Example

First we will start with an example, and explain later on each part of the WSDL. In figures 1 and 2 in pages 15 and 16 you can see a graphical overview of each of the parts we will describe. The WSDL which I'm using for the example is JasparDB [28] [1].

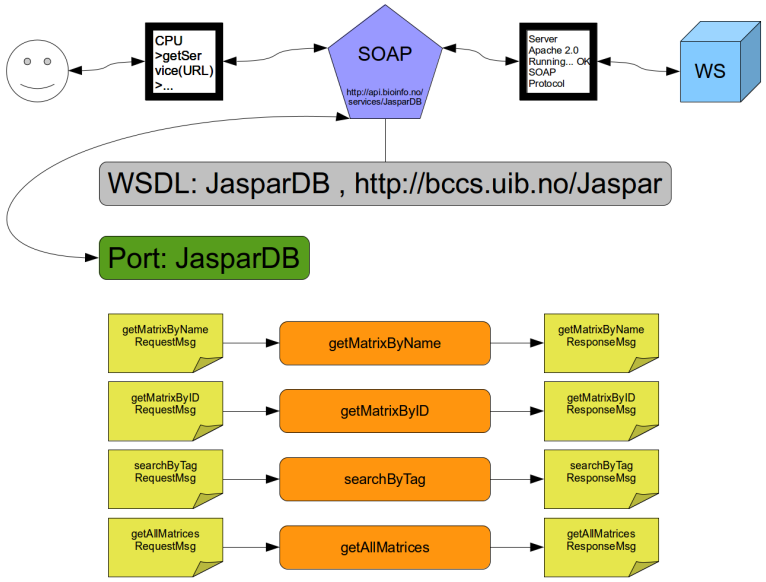

Figure 1: A graphical overview of a WSDL file simplified.

On top left, a representation of a user fetching data from a computer. The computer connect to the WS using a SOAP binding address described in the WSDL file. The WSDL file is represented below the SOAP pentagon, while the server computer and the actual functionality is drawn to the right. The server communicates with the WS in a way that is hidden to the user. The user can only access the functionality described as in the WSDL file. But how the server actually runs the program is concealed to the user. In grey, the targetNamespace given in the definitions section on where you can find this service. The SOAP binding allows to communicate with one port (green) which have many operations associated. Each operation is represented in orange. Each operation may have an input and may also have an output. Input and output communicate with messages represented by a yellow paper. Each operation can have only one input and output, but a message can have many parts coming in or out; but in this case the programmer decided to encapsulate all the information in only one part.

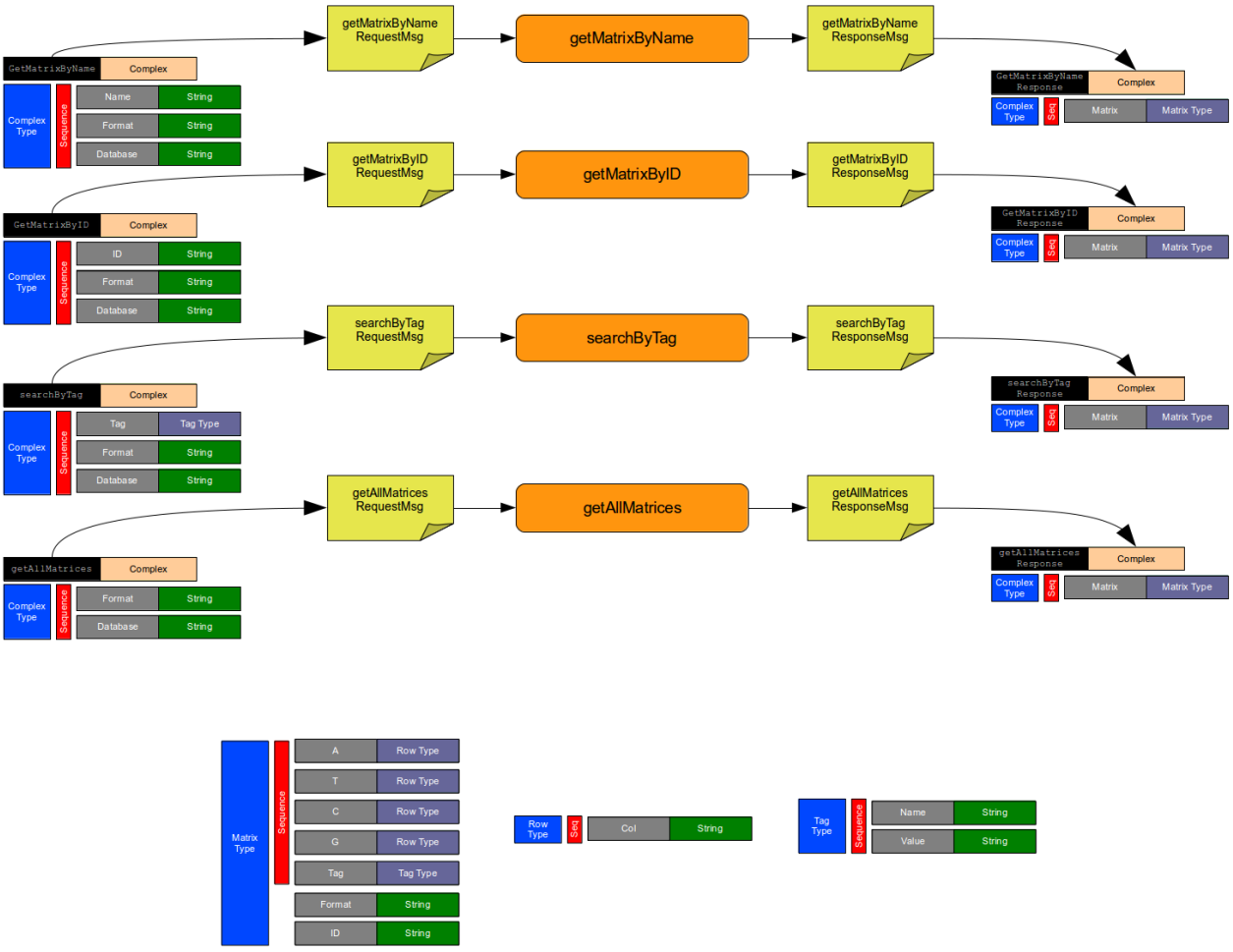

Figure 2: A complete graphical representation of a port in a WSDL file.

This expand the simple view of figure 1. Each complex element have associated a complex type represented in blue. Each complex type is made of several sequences represented with a red stripe, and maybe and some simple elements; in this regard, the W3 foundation recommends not to use sequences but ultimately allows it as part of the standard. In this example there is only one sequence in each complex type. Each sequence is composed by one or more elements. Each element is divided in an element name, in grey, and an element type. If the type is a simple data type (integer, float, boolean, string, ...) is represented with a green background. If it is a complex type is represented with a purple background. Every complex type is described either in the WSDL or in the schemas associated. In this case we have three complex types, Matrix Type, Row Type, and Tag Type which are not associated with any operation but which are there described in the WSDL. At the end you can simplify everything saying that a complex element is decomposed in many simple elements, although in reality there could be many types and may elements in the middle.

In the following lines we will see an example of parts of a WSDL file.

Definitions

This is the definitions part of the WSDL file.

```
<definitionstargetNamespace="http://bccs.uib.no/Jaspar"
xmlns:jas="http://bccs.uib.no/Jaspar"
xmlns:xs="http://www.w3.org/2001/XMLSchema"
xmlns:http="http://schemas.xmlsoap.org/wsdl/http/"
xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
xmlns="http://schemas.xmlsoap.org/wsdl/">
```

We can find the target where the server which communicates with the WS actually is. Also some schemas that describe further the different types used after this section.

Types

This is the element part of the WSDL file. Here is were you can find the schema references in the definition section; and the descriptions of the elements and types as well. The datatypes used here are Position Specific Frequency Matrices (PSFM) [29] which are sufficiently complex to see the recursion relationships of Types and Elements. In this case a PSFM is a matrix of integers which represent the count of nucleotides in their motif position (a motif is a regular expression for nucleotides and amino acids).

```
<types>
<xs:schematargetNamespace="http://bccs.uib.no/Jaspar"
    elementFormDefault="qualified"attributeFormDefault=
    "unqualified" xmlns:ns1="http://schemas.xmlsoap.org
    /soap/encoding/" >

        <xs:element>name="Matrix"type="jas:MatrixType">
        </xs:element>

        <xs:complexTypename="MatrixType">

                <xs:sequence>
                        <xs:element>name="A"type="jas:
                            RowType"> </xs:element>
                        </xs:element>name="T"type="jas:
                            RowType"> </xs:element>
                        <xs:element>name="C"type="jas:
                            RowType"> </xs:element>
                        <xs:element> name="G"type="jas:
                            RowType"> </xs:element>
                        <xs:element> name="Tag"type="jas:
                            TagType"maxOccurs="unbounded">
                             </xs:element>
                </xs:sequence>
```

```
                <xs:attributename="Format"type="xs:string
                    "use="required"> </xs:attribute>
                <xs:attributename="ID"type="xs:string"use
                    ="required"> </xs:attribute>

            </xs:complexType>

            <xs:complexTypename="RowType">
                <xs:sequence>
                    <xs:elementname="col"type="xs:
                        string"maxOccurs="unbounded">
                        </xs:element>
                </xs:sequence>
            </xs:complexType>

            <xs:complexTypename="TagType">
                <xs:sequence>
                <xs:elementname="Name"type="xs:string"> <
                    /xs:element>
                <xs:elementname="Value"type="xs:string"
                    maxOccurs="unbounded"> </xs:element>
                </xs:sequence>
            </xs:complexType>
            ...
            <xs:elementname="getAllMatricesResponse">
                <xs:complexType>
                    <xs:sequence>
                        <xs:elementref="Matrix"
                            maxOccurs="unbounded"/
                            >
                    </xs:sequence>
                </xs:complexType>
            </xs:element>
        </xs:schema>
        </types>
```

You can appreciate the XML usage, as you can see, tags are nested inside each others forming groups. For example; the last element getAllMatricesResponse is made of a complexType which is made of a sequence of only one element, which is an element of type reference, meaning that is not defined there but in other part of the file as the name of "Matrix". If we look back we can find that a Matrix is made of a MatrixType. A MatrixType is defined as a sequence of five elements, A,C,T and G of type RowType and a Tag of type TagType; plus two simple elements Format and ID which are a string each. To properly finish describing this we should search for RowType and TagType. The first is made of a sequence which is made of a lonely element named 'col' of type string. The second is compose by a sequence of two elements, name as a string and value also as a string. Associated with that element we can find useful information, as for example how many of these elements are allowed, if the element is

mandatory or not, which kind of restrictions a string can have, and many more as described in the W3C documentation.

Messages

Here we present the messages section of the WSDL file.

```
<messagename="getMatrixByNameResponseMsg">
  <partname="parameters"element="jas:
      getMatrixByNameResponse"/>
</message>
<messagename="getMatrixByNameRequestMsg">
  <partname="parameters"element="jas:getMatrixByName"/>
</message>
<messagename="searchByTagRequestMsg">
  <partname="parameters"element="jas:searchByTag"/>
</message>
<messagename="searchByTagResponseMsg">
  <partname="parameters"element="jas:
      searchByTagResponse"/>
</message>
<messagename="getAllMatricesRequestMsg">
  <partname="parameters"element="jas:getAllMatrices"/>
</message>
<messagename="getAllMatricesResponseMsg">
  <partname="parameters"element="jas:
      getAllMatricesResponse"/>
</message>
<messagename="getMatrixByIdRequestMsg">
  <partname="parameters"element="jas:getMatrixById"/>
</message>
<messagename="getMatrixByIdResponseMsg">
  <partname="parameters"element="jas:
      getMatrixByIdResponse"/>
</message>
```

Each message has a name and one or more parts. In this example there is only one part for each message with a name and the element associated.

Ports

Here we find the port section of the WSDL file. Each port describe the operations with it.

```
<portTypename="JasparDB">
  <operationname="getMatrixByName">
    <inputmessage="jas:getMatrixByNameRequestMsg"/>
    <outputmessage="jas:getMatrixByNameResponseMsg"/>
  </operation>

  <operationname="getMatrixById">
```

```
    <inputmessage="jas:getMatrixByIdRequestMsg"/>
    <outputmessage="jas:getMatrixByIdResponseMsg"/>
  </operation>

  <operationname="searchByTag">
    <inputmessage="jas:searchByTagRequestMsg"/>
    <outputmessage="jas:searchByTagResponseMsg"/>
  </operation>

  <operationname="getAllMatrices">
    <inputmessage="jas:getAllMatricesRequestMsg"/>
    <outputmessage="jas:getAllMatricesResponseMsg"/>
  </operation>

</portType>
```

Each WSDL have typically one port only. A port have a name and have many operations inside. Each operation is described with a name and may have a message associated with the input and may have one message associated with the output. So at the end if you want to know what elements goes inside an operation you have to follow the thread of messages, parts, element for each part, type for each element, element for each type, and so on.

Bindings

This is the binding section where the SOAP bindings are described.

```
<bindingname="JasparDB"type="jas:JasparDB">
  <soap:bindingstyle="document"transport="http://
      schemas.xmlsoap.org/soap/http"/>
  <operationname="getMatrixByName">
    <soap:operationsoapAction="http://bccs.uib.no/
        Jaspar/getMatrixByName"style="document"/>
    <input>
      <soap:bodyuse="literal"/>
    </input>
    <output>
      <soap:bodyuse="literal"/>
    </output>
  </operation>
  <operationname="getMatrixById">
  <soap:operationsoapAction="http://bccs.uib.no/Jaspar/
      getMatrixById"style="document"/>
    <input>
      <soap:bodyuse="literal"/>
    </input>
    <output>
      <soap:bodyuse="literal"/>
    </output>
  </operation>
```

```
<operationname="searchByTag">
  <soap:operationsoapAction="http://bccs.uib.no/
      Jaspar/searchByTag"style="document"/>
  <input>
    <soap:bodyuse="literal"/>
  </input>
  <output>
    <soap:bodyuse="literal"/>
  </output>
</operation>
<operationname="getAllMatrices">
  <soap:operationsoapAction="http://bccs.uib.no/
      Jaspar/getAllMatrices"style="document"/>
  <input>
    <soap:bodyuse="literal"/>
  </input>
  <output>
    <soap:bodyuse="literal"/>
  </output>
</operation>
</binding>
```

In here is described how to communicate with each operation once you know what is suppose to be sending or receiving from it. For the aim of this thesis this part is irrelevant.

Services

For the last, but not less important, there is the service section.

```
<servicename="JasparDB">

  <doc:ServiceDocumentationxsi:schemaLocation="http://
      www.bccs.uib.no/ServiceDocumentation http://api.
      bioinfo.no/schema/ServiceDocumentation.xsd"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:doc="http://www.bccs.uib.no/
      ServiceDocumentation">

    <doc:WebSite>http://jaspar.genereg.net/</doc:
        WebSite>
    <doc:SampleClient>http://api.bioinfo.no/clients/
        JasparClient.java</doc:SampleClient>
    <doc:SampleClient>http://api.bioinfo.no/clients/
        JasparClient.py</doc:SampleClient>
    <doc:SampleClient>http://api.bioinfo.no/clients/
        JasparClient.pl</doc:SampleClient>
    <doc:Version>0.2 (beta service)</doc:Version>

  </doc:ServiceDocumentation>
```

```
    <portname="JasparDB"binding="jas:JasparDB">
      <soap:addresslocation="http://api.bioinfo.no/
          services/JasparDB"/>
    </port>
  </service>
```

Here you will find the actual name of the WS. Some clients and webs where you can find it. Documentation, metadata; and for us the most important which is the list of ports this WS have. Again; typically only one port per WS is what is most desirable.

Now that we have seen an example for each section lets describe each one of them.

8.4.2 WSDL

The WSDL itself is composed of a set of services and a set of definitions. The services describe the operations, inputs, outputs, way of communication and so on; while the definitions describe datatypes used inside the WSDL and basic information about the WSDL (as for example the name).

Although you can define as many services you want in each WSDL, the logical way of doing it is to define only one since we want our WS to be as modular as possible (which happen to be one of the most important principles of programming). Now; inside the service you will find the relationship between the "ports" and the protocol that you must use in order to communicate with them. I will talk further about the protocols and the ports in section 8.4.7 in pages 25.

Talking about the definitions we can mention three important elements. The name of the WSDL, where is the actual WS, and some more definitions. This definitions are no other things that a namespace where you can find further information about the protocols, types, schemas, semantics and so on. These definitions are also unique identifiers to avoid colliding name terms. For example my Sequence definition could be made by one letter amino acids symbol while your Sequence definition could be made by nucleotides. So in my namespace you will find what is a sequence for me and it will be use to identify my sequences in the documents later on.

8.4.3 Types

This is the most difficult part to understand in a WSDL.

The types are a collection of elements and types. The elements are use by the operations as input or outputs. Every element will have a type associated. For example we can have the element named "Sequence" which have a type "String".

An element won't be as simple as one of the typical elemental data types such integers, strings, boolean, binary, hexadecimal and so on in most of the cases. An element usually will be compose by many elements inside. Sometimes you will have to choose one and only one element from a list to use in your program. An element can have other elements inside it by making use of a Complex Type or a Simple Type. This Complex or Simple Type will have other elements defined inside in various ways which can also contain Complex or Simple Type with more elements until infinity. Because of this recursive relationship it is difficult to visualize the relationship between them.

Elements

In here you will find a list with the kind elements used in WSDLs files. The W3C does not make this distinction. I separated the different kind of elements to help the reader to understand the WSDL file.

An element have a name, this in one of the two attributes which all these kind of elements have in common. The other thing that all of them can have is a list of semantic annotations, which we will see deeper in detail in chapter 8.6 in page 28.

inLine Elements

These kind of elements are defined in a single line in the WSDL file, hence the name. They are defined by a name and a type (which remember, could be elemental datatype, complex or simple). They also could be fixed to a given value defined in the file. This kind of element is the lower level of recursion possible, and at the end, every other element, no matter how complex it is, must be defined by a lot of inLine elements which have a type that is an elemental datatype.

References Elements

These elements are just pointers that refer to another element defined before in the file. So instead of copy and pasting the part of the file that define that element (which would also work) you just write this pointer, and the program that process this WSDL should be clever enough to interpret it and understand to which element is actually referring to.

Choice Elements

These kind of elements have a set of elements defined inside them, usually inLine Elements. What is different from Complex Elements is that you can only use one of the elements defined inside, and you must choose one of them. For example in a BLAST search you could search by element Amino acid Sequence or by element Nucleotides Sequence, but you can't mix them or not use any of them; you must select one.

Simple Elements

This element has a Simple Type defined inside it. You will see later what is a Simple Type; but basically it has a type and a set of rules associated such as "only A, T, C, G characters" or "smaller than three" for example.

Complex Elements

This element have a list of Complex Type defined inside it, although it will be usually only one Complex Type.

Types

Here I'm going to describe the types. The name is unfortunate because they are inside the section of types in a WSDL; so sometimes it could be confusing to understand if we are talking about the types of a WSDL or the types defined inside the Types section of a WSDL which are use by the elements. Here again, both simple and complex types can have a list of semantics annotations, and they could also have a name, but this is not mandatory if they are defined inside an element.

Simple Types

This type has a name and a base associated. The base is just another name for defining a type of variable. So a base would be something like String, Integer and so on. The name of the Simple Type is just an identifier and not important at all since the base is what define the kind of type is the Simple Type. The simple Type can have a list of restrictions defined which are a set of options from where you have to pick one to give as value to the base. Again with the BLAST example, a simple Type could be defined with name equal to "Sequence Type", base as "String" and the restrictions would be "Amino acid" , "DNA", "Amino acid-Three Letters" and "RNA" for example. Each of these enumerations can have a semantic annotation associated.

Complex Types

This last type is defined sometimes by a name, a list of sequences and a list of attributes. A sequence is a list of elements given in an order and the complex type can have many sequences, although usually is only one. An attribute is a special kind of element but it behaves exactly as an inLine Element. They are special because how the XML document that you gives to the WS is formatted. The W3C actually discourages their use. In figure 2 in page 16 we can see that the programmer decided to use Complex Types nevertheless.

One last thing to comment about the type section in a WSDL. When you find the type you will find it with the proper namespace that I mentioned before. So instead of "String" you will actually find something like

"xs:String" and in the definitions of the WSDL what does it means the prefix "xs" and how the type "String" is defined in there.

8.4.4 Messages

A message is what you give to an operation or what the operation gives back to you. It has a name that identifies it uniquely. Beside that it has a list of part and each parts has an element from the Type section associate to it.

Lets suppose we have a multiplication operation. It needs two numbers as input. You could define as input for that operation a Message which has two parts and each part is associate with the element "Number". Or you could have a Message with only one part which is associate with the element called "Number_pair" which inside has a Complex Type with two elements "Number" inside.

8.4.5 Ports

A port is a collection of operations. A WSDL can have several ports that shares operations between them but again it would be more logical to create another WSDL file to keep the principle of modularity. The port have a name as unique ID and can have a list of semantics associated.

8.4.6 Operations

An operation is exactly what it sounds. A function that can have an input and can have an output. They could also have a "failure" but this is not important for the scope of this thesis. Both input and output receive or gives a Message. The operation have a name and could have a list of semantics related.

8.4.7 Bindings

In this section is described how to communicate with the WS. Each port has a binding and each binding can have a set of protocols associated; usually SOAP. For each protocol you have a set of operations and how to communicate with them. In my application I'm going to ignore any binding restrictions and I'm going to assume that every given two WS have some sort of universal communicator that allow then to talk freely without human intervention.

8.5 EDAM Ontology

An ontology is a relationship between concepts[30]. There are many types of ontologies that focus on a particular scenario. In here I'm going to use the EDAM ontology [21] that describe relationships between biology concepts oriented to the web services.

8.5.1 Sub-ontologies

The ontology consist on 5 main sub-ontologies called "Data", "Format", "Operation", "Identifiers" and "Topic". The ontologies are setup in a non-cyclic graph and each node of the graph represent some abstract concept inside that ontology. For example "Sequence" node in Data subontology, "Sequence Alignment

Construction" in Operation subontology, and so on. All subontologies have the common property of having abstract concepts at the top levels while having more detailed and specific examples at bottom levels.

8.5.2 Relationships

Subontologies can have relationships with each other or with themselves. The different relationships that are in EDAM are the following:

HAS_INPUT/HAS_OUTPUT:
This defines a relationship between an operation element and a data element. For example, the operation element Sequence Alignment Analysis has a relationship HAS_INPUT with the data element Sequence Alignment.

IS_A:
This define an specialization between A and B. For example, a phylogenetic tree construction is an analysis operation (specialization), but not all analysis operations are about phylogenetic trees; we can have for example analysis of structures like finding cavities. The relationship is transitive. If A is a B and B is a C, then A is a C. The transitivity still applies for the has_input/has_output relationship. For example, if A has input B and B is a C, then A can accept both inputs B or C. This relationship only apply with elements of the same subontolgy. Operation can be a generalization of another Operation but not a generalization of a Data element. In figure 3 we see an example of this relationship.

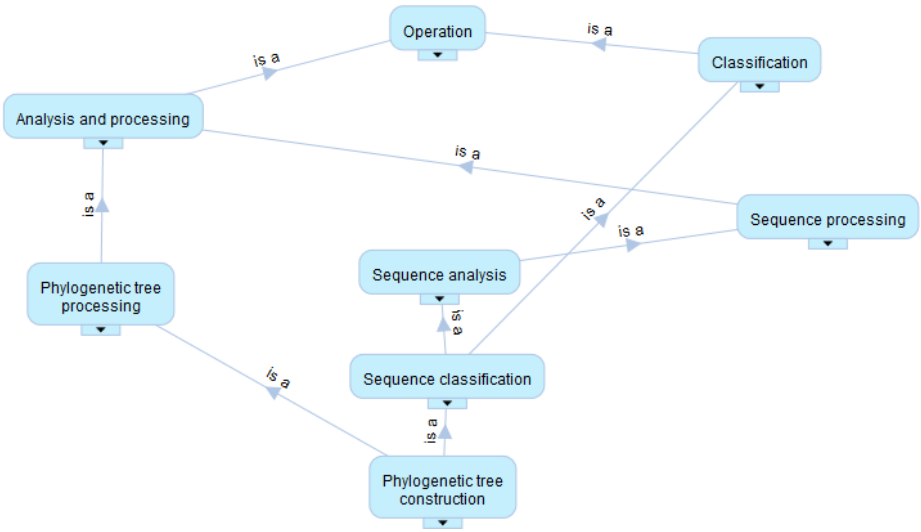

Figure 3: An overview of the path of Phylogenetic Tree Construction to the root, using the is_a relationship.

In this figure we can see several elements of the subontology Operation. Each element is represented by a rectangle with round corners with a light blue background. At the bottom of the figure we find the Phylogenetic Tree element. At the top, the Operation element. In between them the different paths that you can follow to reach the Operation element from the Phylogenetic Tree element using the is_a relationship. This image was acquired using a visualization tool for ontologies at `http://bioportal.bioontology.org/ontologies/EDAM/?p=summary`.

HAS_TOPIC:
: This is a relationship between Data and Topic or Operation and Topic. If A has topic B, it means that A is somehow included in the scope of Topic B. For example, the operation Protein model evaluation has the topics Homology modeling, Protein tertiary structure prediction, and Molecular modeling

IS_IDENTIFIER_OF
: A relation between an Identifier and a Data. In the context of this thesis, this relationship is not in use.

IS_FORMAT_OF
: Designate a Format for a Data element. This specifies the different kinds of formats that a Data element can have. Be aware that this doesn't tell which Format the Data actually has. So for example, we can have a sequence of amino acids like AVLI... , and another sequence like Ala, Val, Leu, Ile... They are exactly the same sequence but defined with different format.

8.6 SAWSDL

This stands for Semantic Annotations for WSDL. A semantic annotation is a link between a part of the WSDL and a semantic concept. The semantic concept is likely to be described somewhere else outside the WSDL and more in particular in an ontology. In our case, we are going to use the concepts from the EDAM ontology but be aware that it is perfectly possible to find links to another ontologies.

In figure 4 in page 28 we can see a graphical example between the relationships inside the sub-ontologies and the relationships from a WSDL.

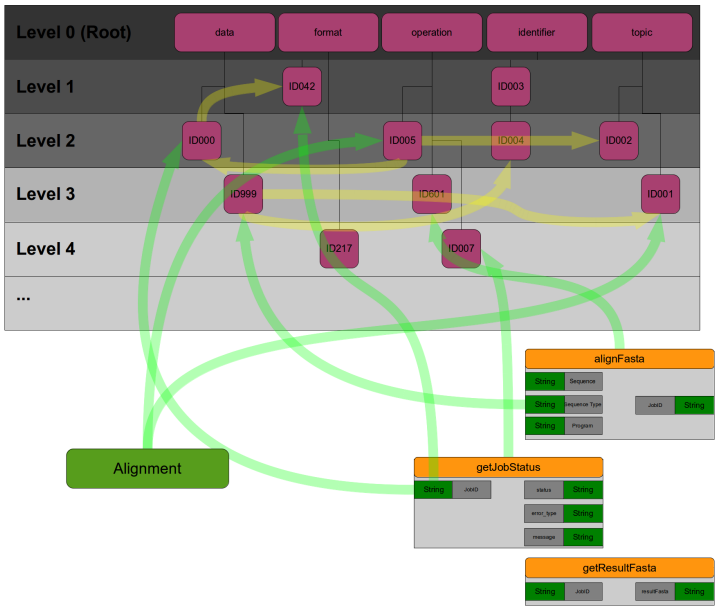

Figure 4: Semantic Relationships between a WSDL and EDAM.

In yellow, links between elements of the ontologies. In green, links from a WSDL to an element of an ontology. Later on in chapters 9.7 and 9.10 in pages 36 and 40 I'm going to expand the idea of why the semantic annotations are important and which kinds of properties I'm going to give to them.

If we want to annotate a concept in a WSDL file we simply add the keyword 'sawsdl:modelReference' to the concept tag. Here is an example of an element sequence being annotated with a Data element and a Format element:

```
<xsd:element name="sequence" type="xsd:string" sawsdl:
    modelReference="http://edamontology.org/data_2043 http://
    edamontology.org/format_2200"/>
```

8.7 CSS3

CSS stand for Cascading Style Sheet. It is a code used to describe the formatting of a document. Normally it is use for describing HTML and XML documents; although there are examples for many other uses [31]. The purposes of CSS is to separate the definition of how the document look from the document itself. In that way you can share documents in a more simple way, or program websites that looks different on a smart phone than in your 24 inches monitor.

CSS is implemented in a series of modules that add functionality and a greater variety of potential uses. There are more options to represent a particular layout in CSS2 than in CSS1 and so on. This program is using CSS3 because at the time of the implementation the documentation for CSS4 was still not released.

8.8 JavaScript

JavaScript is an interpreted programming language [32] , this means the code is not compiled but executed as it is needed. It is use by a web browser to execute code in the client-side and create anything that the program would do, such a games, media players, office applications and many different types of programs.

8.9 HTML5

This is just the combination of HTML + CSS + Javascript. So if I say that the program is implemented in HTML5 is means that there are pieces of the program in each of these sections.

8.10 Acid Test

Acid Test is a test done in web browsers to grade how well they are handling web standards. It is particularly emphasized in CSS and JavaScript [33]. The last version is the Acid3. I will make a brief reference to this later on speaking about WSDL-Workshop in section 10.3.1 in page 49.

8.11 Summary

Here so far we have reviewed every concept which you need to know in order understand the more creative part of the thesis, which follows in the next section.

9 Mathematical model for web services

The overview of this thesis is that we should connect several web services in some sort of chain. A web service can connect to several others web services, and at the same time several other can connect with it.

When a web service connects to another we say that at least one output of a WS connects with an input of a WS. We will call these WS_{Out} for the one that gives outputs and WS_{In} for the one that receive them.

A random WS_{Out} can connect to a WS_{In} if the type of input match the type of output. That will make sense to connect from the technical point of view; as in when you give a string to a function that requires a string. However there are more than only that.

A human could randomly connect WS's and at the end get an output, although this output will lack of any kind meaning. We would like to be able to connect to WS's that makes sense to connect. So we could start by connecting a WS_{Out} to a WS_{In} which has the same kind of data; not in the sense of elemental datatypes (such us integer, float, char, etc...) but connecting WS_{Out} that gives a peptide to a WS_{In} that receive a peptide, independently from the datatypes used to describe the peptides.

How to achieve this? That is where the semantics come into play. If WS_{Out} has a semantic annotation that points to, for example, peptide, and the WS_{In} has a semantic annotation that point to peptide, this means that the WS's have make sense to connect.

Now that I have presented the general idea behind this, let's see each property in detail. First I'm going to show the state of the art, and show current technologies that make WS to talk to each other. Here we will see that the current technology is not good enough, specially using scientific WS, although the use of commercial WS is promising. Since none of them provide the tools needed, I will then proceed to describe my own hypothesis. In the next chapter we will see the program that test that hypothesis. And in the next one the results.

9.1 Previous comments on technologies

Currently there are many many technologies that works with WSDLs or other concepts which are similar to WSDLs. Some of them are available right now, some of then are in development and some of then are just concepts in early stages. And of course many of then are not even free-software. For every technology that you investigate you will find references to other technologies which you haven't hear before. So the first challenge was to identify something that was able to help with task at hand.

We want a workbench with nice functionality like Taverna or Galaxy. And we want it to be able to analyze the semantics between services but the current technologies neglect the semantic part of the WSDLs and the only alternative, which is SWWS (below) doesn't exist yet. This is an overview of the most popular options:

9.1.1 OWL-S Semantic Markup for Web Services - The OWL Services Coalition

OWL-S is an ontology with an extension for declaring abstract WS [34] [35]. You will construct such abstract WS, and make the connections between them at a high level of abstraction. Later on you will have to find a WSDL that fits your abstract WS.

9.1.2 The Web Service Modeling Framework WSMF

WSMF describe a theoretical solution for businesses WS [36] [35]. The definition speaks about four essential entities, that working in common, will make a better overall semantic web state. First ontologies, giving approximately the same use as in here or other technologies. Then we have goal repositories, which is just the functionality for each WS, but not for a workflow. Furthermore, talks about Web Services and how to connect them to each other at low level to make the data flow. And for the last it describe the concept of mediator and discuss several cases. A mediator is something I will later call translator, and is a piece of software which will communicate to WS which are in principle compatible but who talk a difference language (use different data structures, use different transmission protocol, and so on).

9.1.3 WSCL 1.0

Web Service Conversation Language [37] was a proposed by the Hewlett-Packard Company to provide an XML file to complement the WSDL file and make a WS from a company be able to interact with the WS from another company.

9.1.4 WS-BPEL

Web Services Business Process Execution Language is a language based on WSDL1.1 to specify WS and for them to interact automatically. It describes two kind of processes. Executable processes, a model of the behavior of a business interaction, and Abstract processes which are just descriptions of the possible use case and are not intended to be executed. What it does is add more code to a WSDL1.1 file with a tag system to model how the WSDL is executed. Examples of these tags at operation level are <receive>, <reply>, <invoke>, <assign>, <throw>, <exit>, <wait>, <empty>, <sequence>, <if>, <while>, <repeatUntil>, <forEach>, <pick>, <flow>, <scope>, <compensate>, <compensateScope>, <rethrow>, <validate>, <extensionActivity>. [38].

9.1.5 SWWS

This is just a framework of the several technologies described before [35] [39]. It stands for Semantic Web enabled Web Services and it is a long term objective to combine the semantic web and web services into Intelligence Web Services. It relies heavily on WSMF and is funded by the European Union. Aim to be the standard in the near future and it is probably the best option in the future. However there is nothing implemented yet and everything is pure theoretical.

9.2 Review of a WSDL structure

In figures 1 and 2 in page 15 and 16 we saw an overview of the different part of a WSDL. I will refer to this image and their shapes and color scheme in future comments to identify different parts of the WSDL. Following, we will see what kind of variables compose an operation. How to link such data in a correct manner. How to link operations in a correct manner. An abstract idea of linking variables and operations. And an approach to differentiate good workflows from irrelevant workflows.

9.3 Flattening operations and shredding elements

We saw in the WSDL description in section 8.4 in page 13 all the components of a WSDL. Now I'm going to simplified the model a little bit by removing superfluous information without loosing important elements. Flattening an operation means to take out the messages and link the elements directly to the input and output. With these we can skip the message component of the operation and refer directly to the inputs and outputs.

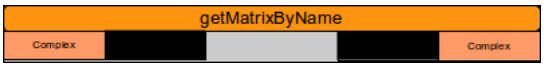

Figure 5: An operation flattened

In here we see an operation with no messages. Instead we took the content of the messages and gave it to the operation to simplify the view. However we haven't gained any useful information because the messages were composed of only one complex type.

Shredding an element means to take out from a complex element the final elements that compose it. For example, an operation usually have only one input element which name is something like "Name of the operation"+"Request" which is made of a complex type with a set of elements. For the user the variable "alignRequest" doesn't have too much sense but it would be more intuitive to have the three elements that "alignRequest" has; which are for example "Sequence A", "Sequence B" and "Algorithm". Shredding can be done recursively until only basic datatypes remain.

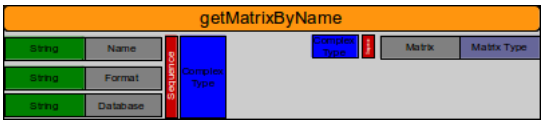

Figure 6: An operation flattened which is also shredded

In this operation we took out the complex type from the message and assigned it to the operation the elements which compose the message; in contrast with giving the complex type with have no information of what is inside.

Flattening a port can only be achieved if a port is suppose to have all its operations correlated. For example you give a sequence to an operation A and the operation gives you a job id. Later on, that job id is given to another operation B in order to collect your results. In that way we can simplify the

port that contains operations A with input A and output A, and operation B with input B and output B to just a port with input A and output B. But this, again, might not be possible.

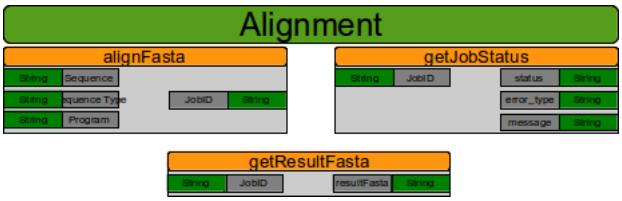

Figure 7: A port which is a candidate to be flattened

In this case you can give the output of alignFasta operation to getResultFasta input; so the port would be flattened.

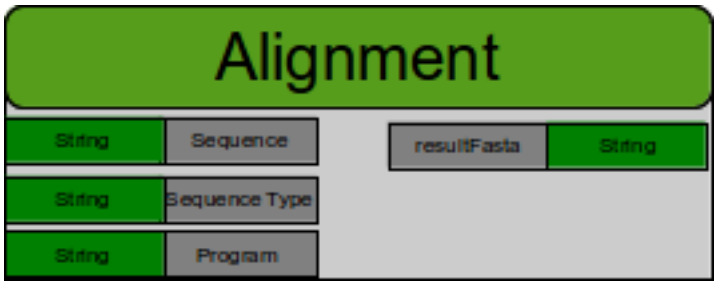

Figure 8: A flattened port

Same port of figure 7 after being flattened.

Flattening an operation and shredding an element is very easy for a computer to do. However is difficult for a given set of operations to check if they have a first and last operation; or if in the contrary the operations are completely unrelated. Here is my suggestion to try to flat ports automatically. Some examples are provided in the description; this examples refer to figure 9 in page 34. All of these conditions must occur at the same time.

A port is candidate to be flattened if:

- Has a semantic link to an operation; lets call this "Link S". (Green Arrow from port to operation ID601 in the figure 9)
- The operation inside the ontology has input and output, and both are internally linked inside the ontology with at least one data; lets call this data D_{IIS} and D_{OIS}. (Yellow arrows from ID601 to ID000, ID026 and ID999)
- The amount of internal links with the input data is defined as Input Multiplicity. Note that an operation may require several inputs of the same data; each one of those would add 1 to the Input Multiplicity. Same idea applies to the output, which will be called Output Multiplicity.

- It has only one candidate to first operation and only one candidate to last operation.

An operation is candidate for first operation if:

- The number of inputs it has match the Input Multiplicity of the port.
- These inputs have a set of semantically annotated links (green arrows) to elements of the Data Ontology. This set must be equal to D_{IIS}.

An operation is candidate for last operation if:

- The number of outputs it has match the Output Multiplicity of the port.
- These outputs have a set of semantically annotated links (green arrows) to elements of the Data Ontology. This set must be equal to D_{OIS}.

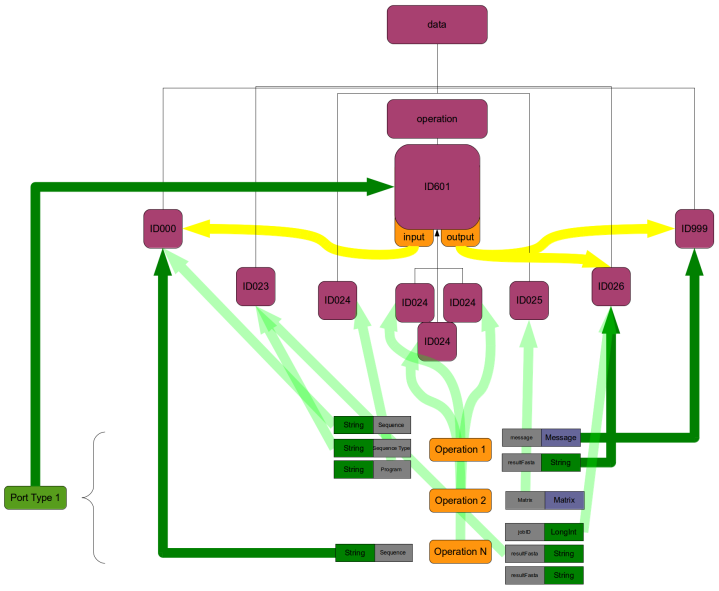

Figure 9: Scheme of a port that can be flattened

In dark green the semantics annotations of the first and last operation. In light green other semantics annotations from other operations that doesn't fit the definition for either first operation or last operation.

In the worse case scenario that the computer fail to achieve this automatically, the user can still link operations manually. So the port operations are executed in the proper order.

9.4 Operation and port duality

An operation have inputs and outputs. A port that has been flattened also have inputs and outputs. In essence a flattened port and an operation is the same

concept, a black box with inputs and outputs that are semantically annotated and are made of different datatypes. So both will follow the same rules that I'm going to explain now.

9.5 Mathematical definitions

Until now we have see diagrams and images to help to understand the situation. Let me define formally some of the elements that I'm going to use later on. Remember that this model is based on the flattening version of a WSDL:

All WSLDs in existence are contained in the set Ξ and each one is denoted uniquely as WSDL_0, WSDL_1, ... , WSDL_N.

A random WSDL_i have a set of Ports noted as Φ_i and each one is denoted uniquely as P_{i1}, P_{i2},... ,P_{iM}. Typically a WSDL will only have one port, but by definition it can have many.

A random P_{ij} have a set of Operations noted as Ω_{ij} and each one is denoted uniquely as O_{ij1}, O_{ij2},..., O_{ijO}. This set is never empty.

The intersection of a pair Ω_{iA} and Ω_{iB} with A $\neq$ B is not necessarily empty.

A random operation O_{ijk} have a set of inputs and outputs noted as O_{ijk}IN and O_{ijk}OUT. These sets can be both empty and by definition of the W3C a WSDL may have no outputs at all. In the case they have elements, each one of them are denoted uniquely as $\text{O}_{ijk}\text{IN}_1$, $\text{O}_{ijk}\text{IN}_2$,..., $\text{O}_{ijk}\text{IN}_X$ and $\text{O}_{ijk}\text{OUT}_1$, $\text{O}_{ijk}\text{OUT}_2$,..., $\text{O}_{ijk}\text{OUT}_Y$ respectively.

A random input or output may be a complex element or a complex type, and thus be discomposed in several parts. If it is complex it will be denoted with a + next to the number that identifies it, like: $\text{O}_{ijk}\text{OUT}_{y+}$. Each one of these parts will be notated with more numbers, as in $\text{O}_{ijk}\text{OUT}_{y1}$, $\text{O}_{ijk}\text{OUT}_{y2}$,... , $\text{O}_{ijk}\text{OUT}_{yn}$. At the same time, each one of these parts can also be a complex part and thus be able to expand further, so they will be denoted again with a + if that is the case and be noted with a / to separate with the rest of the identifier if we want to write it in an expanded form. For example: $\text{O}_{ijk}\text{OUT}_{y1+}$, $\text{O}_{ijk}\text{OUT}_{y2/1}$, $\text{O}_{ijk}\text{OUT}_{y2/2+}$, $\text{O}_{ijk}\text{OUT}_{y2/3/1+}$,$\text{O}_{ijk}\text{OUT}_{y2/3/2}$, $\text{O}_{ijk}\text{OUT}_{y2/3/3}$, $\text{O}_{ijk}\text{OUT}_{y2/4}$, $\text{O}_{ijk}\text{OUT}_{y3}$, ..., $\text{O}_{ijk}\text{OUT}_{yn}$.

A random input or output may be a choice element. It will have a list of elements (complex or simple) from which you must choose one. In this case we will denote it with a 'c' next to name in order to indicate that is a choice element, and in between '{}' the possible choices if we want to expand the view: $\text{O}_{ijk}\text{OUT}_{y1+}$, $\text{O}_{ijk}\text{OUT}_{y2+}$, $\text{O}_{ijk}\text{OUT}_{y3}$, $\text{O}_{ijk}\text{OUT}_{y4c}$, $\text{O}_{ijk}\text{OUT}_{y5\{1,2+,3/1,3/2c,3/3+\}}$, ... , $\text{O}_{ijk}\text{OUT}_{yn}$.

A random input or output have a type associated. This is represented by ':' next to the final character of the input or output. However choice elements can't have this notation unless they are expanded, because a choice element don't have any type by itself, the choices are the one with the properties: $\text{O}_{ijk}\text{OUT}_{y1+:matrix}$, $\text{O}_{ijk}\text{OUT}_{y2+:ID}$, $\text{O}_{ijk}\text{OUT}_{y3/1:string}$, $\text{O}_{ijk}\text{OUT}_{y3/2:int}$, $\text{O}_{ijk}\text{OUT}_{y3/3:float}$, $\text{O}_{ijk}\text{OUT}_{y4c}$, $\text{O}_{ijk}\text{OUT}_{y5\{1:int,2+:matrix,3/1:string,3/2c,3/3+:row\}}$

, ... , $O_{ijk}OUT_{yn:PBD}$.

Finally; any random input or output, any operation, or any port, have a set of semantic annotations which can be empty. The set is notated by the name of the concept plus Σ and the elements are labeled as S_1, S_2,... , S_P. If we want to refer to a particular subset of the semantics, we notated as Σ(<list of names>) , as for example Σ(data, operation) to refer to only semantics that belongs to the sub-ontologies Data and Operation. In the case of the operation we can refer to the "has_input" or "has_output" semantics conditioning the data, like as O_{ijk} Σ(data|input).

9.6 Inheritance property in a WSDL

P_{ij} Σ have its elements copied to every Ω_{ij} Σ.

If a $O_{ijk}\Sigma$ contains an operation element A which has a relationship of 'is_input' to an element B, then, either there is at least one element in O_{ijk}IN which have an element in $O_{ijk}IN\Sigma$ equal to B or there is one that can potentially be linked to B. This property works as well for the outputs counterpart.

9.7 Making links between an input and an output

This section will talk about the possibility of linking datatypes and won't take into account the semantics, which will be described later. Two consecutive operations could be semantically compatible, but has no compatible inputs/outputs.

Given two random operations O_{ijk} and O_{xyz}, we can describe how compatible are the output from the first operation $O_{ijk}OUT_A$ with the input of the second operation $O_{xyz}IN_B$.

There are three things to consider here. The type, the semantics with the data ontology, and the semantics with the format ontology.

The type alone is not good enough to evaluate how compatible they are. If we have a String with a String, sure, that is easy and they are probably compatible. What about a 'matrix' with 'matrix'? What about 'int' with 'integer'? Or 'float' with 'int'? Or maybe the WSDLs are even in different languages and the programmer set up the datatype to his mother language and forgot to change it to English, or have a typo, or who knows. I'm going to say that a datatype is the same of other datatype if the string that identify the type is the same and if the schema that define the datatypes are the same. The list of elemental datatypes are defined by the W3C and you can find them here: "duration", "dateTime", "time", "date", "gYearMonth", "gYear", "gMonthDay", "gDay", "gMonth", "boolean", "base64Binary", "hexBinary", "float", "double", "anyURI", "QName", "NOTATION", "string", "decimal", "normalizedString", "integer", "token", "nonPositiveInteger", "long", "nonNegativeInteger", "language", "Name", "NMTOKEN", "negativeInteger", "int", "unsignedLong", "positiveInteger", "NCName", "NMTOKENS", "short", "unsignedInt", "byte", "unsignedShort", "unsignedByte". So for example xs:string is equal to xsd:string if 'xs:' and 'xsd:' have the same definitions.

Let's call the datatype of the output DT_A and the datatype for the input DT_B, and lets take a look to every possible combination. Each combination has an identifier in the form of [i] which I will reference later on. I'm setting a tree model. For [i] to be true, all conditions of each of the branches that leads to [i] must happen at the same time.

- $\mathrm{DT}_A = \mathrm{DT}_B$
  - $\mathrm{O}_{ijk}\mathrm{OUT}_A\Sigma(\text{data}) \neq \varnothing$ and $\mathrm{O}_{xyz}\mathrm{IN}_B\Sigma(\text{data}) \neq \varnothing$.
    - $\mathrm{O}_{ijk}\mathrm{OUT}_A\Sigma(\text{data}) \cap \mathrm{O}_{xyz}\mathrm{IN}_B\Sigma(\text{data}) = \mathrm{O}_{ijk}\mathrm{OUT}_A\Sigma(\text{data})$
      - $\mathrm{O}_{ijk}\mathrm{OUT}_A\Sigma(\text{format}) \neq \varnothing$ and $\mathrm{O}_{xyz}\mathrm{IN}_B\Sigma(\text{format}) \neq \varnothing$.
        - $\mathrm{O}_{ijk}\mathrm{OUT}_A\Sigma(\text{format}) = \mathrm{O}_{xyz}\mathrm{IN}_B\Sigma(\text{format})$ [1] As equal as it can get. We have a complete description of the ontology for all the pieces. Everything match, no doubt that this must be possible to link with each other.
        - $\mathrm{O}_{ijk}\mathrm{OUT}_A\Sigma(\text{format}) \neq \mathrm{O}_{xyz}\mathrm{IN}_B\Sigma(\text{format})$ [2] Everything else match except the format. For example, my aminoacids are described as ACT,GTA,GGC,... while yours are described as Ala,Gly,Ser.
      - $\mathrm{O}_{ijk}\mathrm{OUT}_A\Sigma(\text{format}) = \varnothing$ or $\mathrm{O}_{xyz}\mathrm{IN}_B\Sigma(\text{format}) = \varnothing$. [3] We know that one aminoacid goes out and one goes in; but we have no idea of the format of one of them.
    - $\mathrm{O}_{ijk}\mathrm{OUT}_A\Sigma(\text{data}) \cap \mathrm{O}_{xyz}\mathrm{IN}_B\Sigma(\text{data}) \neq \mathrm{O}_{ijk}\mathrm{OUT}_A\Sigma(\text{data})$ and $|\mathrm{O}_{ijk}\mathrm{OUT}_A\Sigma(\text{data}) \cap \mathrm{O}_{xyz}\mathrm{IN}_B\Sigma(\text{data})| \geq 1$ [4] Among other elements that doesn't match, there is at least one element in common between the input and the output in the semantic set. We cannot say anything about the format, because the format references are mixed and could refer to other data annotation which is not the one they have in common.
    - $\mathrm{O}_{ijk}\mathrm{OUT}_A\Sigma(\text{data}) \cap \mathrm{O}_{xyz}\mathrm{IN}_B\Sigma(\text{data}) = \varnothing$
      - DT_A and DT_B are elemental datatypes.
        - $\mathrm{O}_{ijk}\mathrm{OUT}_A\Sigma(\text{format}) = \mathrm{O}_{xyz}\mathrm{IN}_B\Sigma(\text{format})$ or $\mathrm{O}_{ijk}\mathrm{OUT}_A\Sigma(\text{format}) = \varnothing$ or $\mathrm{O}_{xyz}\mathrm{IN}_B\Sigma(\text{format}) = \varnothing$ [5] There are semantic annotations to data and they are all wrong. However is an elemental datatype. This case should be impossible. It is like trying to describe a car license plate the same way you describe a street address.
        - $\mathrm{O}_{ijk}\mathrm{OUT}_A\Sigma(\text{format}) \neq \mathrm{O}_{xyz}\mathrm{IN}_B\Sigma(\text{format})$ [6] They are elemental datatypes but the semantics and formats have nothing in common.
      - DT_A or DT_B is not elemental datatypes. [7] Although the datatypes have the same name, they are complex datatypes and is difficult to say that they are the same because the semantic they have doesn't fit at all. In this case I'm going to say that they are not compatible and they can't be linked.
  - $\mathrm{O}_{ijk}\mathrm{OUT}_A\Sigma(\text{data}) = \varnothing$ or $\mathrm{O}_{xyz}\mathrm{IN}_B\Sigma(\text{data}) = \varnothing$.
    - $\mathrm{O}_{ijk}\mathrm{OUT}_A\Sigma(\text{format}) = \mathrm{O}_{xyz}\mathrm{IN}_B\Sigma(\text{format})$ [8] We don't have semantic data, but for some reason the format is in there and it is the same. So we are good and they can be linked.

    * $O_{ijk}OUT_A\Sigma(\text{format}) \neq O_{xyz}IN_B\Sigma(\text{format})$ [9] We don't have semantic data and the format is different.

- $DT_A \neq DT_B$
  - $O_{ijk}OUT_A\Sigma(\text{data}) \neq \varnothing$ and $O_{xyz}IN_B\Sigma(\text{data}) \neq \varnothing$.
    * $O_{ijk}OUT_A\Sigma(\text{data}) \cap O_{xyz}IN_B\Sigma(\text{data}) = O_{ijk}OUT_A\Sigma(\text{data})$
      · $O_{ijk}OUT_A\Sigma(\text{format}) \neq \varnothing$ and $O_{xyz}IN_B\Sigma(\text{format}) \neq \varnothing$.
        $O_{ijk}OUT_A\Sigma(\text{format}) = O_{xyz}IN_B\Sigma(\text{format})$ [10] They don't have the same name but they match in everything else. My guess is that this is impossible for elemental datatypes (otherwise they won't have the same format). This is a case of two different people programing the same kind of variable but with different datatype name.

        $O_{ijk}OUT_A\Sigma(\text{format}) \neq O_{xyz}IN_B\Sigma(\text{format})$ [11] Same as before. Probably impossible for elemental datatypes. This concept output match the input of the other one even though they have different formats; so is just a problem of translation between data.
      · $O_{ijk}OUT_A\Sigma(\text{format}) = \varnothing$ or $O_{xyz}IN_B\Sigma(\text{format}) = \varnothing$. [12] We know that there is something similar going on but we are unsure of the specification of the format.
    * $O_{ijk}OUT_A\Sigma(\text{data}) \cap O_{xyz}IN_B\Sigma(\text{data}) \neq O_{ijk}OUT_A\Sigma(\text{data})$ and $|O_{ijk}OUT_A\Sigma(\text{data}) \cap O_{xyz}IN_B\Sigma(\text{data})| \geq 1$ [13] Among other elements that doesn't match, there is at least one element in common between the input and the output in the semantic set. This is actually a poor possible link, you would have to take a look at this manually to see if it can be fixed to work somehow.
    * $O_{ijk}OUT_A\Sigma(\text{data}) \cap O_{xyz}IN_B\Sigma(\text{data}) = \varnothing$
      · DT_A and DT_B are elemental datatypes.
        $O_{ijk}OUT_A\Sigma(\text{format}) = O_{xyz}IN_B\Sigma(\text{format})$ or $O_{ijk}OUT_A\Sigma(\text{format}) = \varnothing$ or $O_{xyz}IN_B\Sigma(\text{format}) = \varnothing$ [14] Don't have the same semantic not even the same name. For an elemental datatypes this is something very bad even if somehow the manage to share the format.

        $O_{ijk}OUT_A\Sigma(\text{format}) \neq O_{xyz}IN_B\Sigma(\text{format})$ [15] This is the bigger difference that you can have. Elemental datatypes with different semantics and different format.
      · DT_A or DT_B is not elemental datatypes. [16] Once again, this is a complex datatype with different semantic and even the format match by miracle they are not remotely the same kind of variable.
  - $O_{ijk}OUT_A\Sigma(\text{data}) = \varnothing$ or $O_{xyz}IN_B\Sigma(\text{data}) = \varnothing$.
    * $O_{ijk}OUT_A\Sigma(\text{format}) = O_{xyz}IN_B\Sigma(\text{format})$ [17] We don't have semantic data, but for some reason the format is there and is the same. The name is different, so it would be a tentative to call this different.

        * $O_{ijk}OUT_A\Sigma(format) \neq O_{xyz}IN_B\Sigma(format)$ [18] We don't have semantic data and the format is different and the name is different. Again a wild guess but this doesn't look similar at all.

Beside all of these there is one more thing to consider. Sometimes a data is a specialization of another data. So even if the semantic link is not equal the semantic data is still valid. For example an operation that accepts an input which is a Sequence could handle amino acids, alignments, and all type of sequences in general. That needs to be taken care of, and when we check if a semantic is equal to other, we need to see if they are not equal they can still be a specialization of the other. For example a protein sequence should be able to link with a sequence; despise that they are not the same element in the data ontology, you can still reach sequence from protein using the "is_a" relationship.

This however only works for outputs which are specializations of the next input. If the input is the specialization of the output, we cannot assure that everything works fine.

We will say that Operation A can link with Operation B if at least one element from the output of A can link with one element from the input of B.

9.8 Links interchangeable property

Lets consider 3 operations which are joined in a workflow called A, B, and C. We say that a forth operation D can be interchange with the middle one B if all the link types between A and B and B and C are the same types of links between A and D and D and C.

9.9 Making links simplified version

In the final implementation I simplified the types of links that you can have. It is quite trivial to expand everything and make all possible 17 combinations; however to have 18 different color scheme for links in between operations is quite overwhelming for the user. So just for a proof of concept demo I reduced all cases to just the following.

In the first place, the format disappears and I'm going to consider that somewhere there is a universal translator between formats. This could actually be a potential master thesis on its own, to make a program that translate in between every possible format that can be converted to another (for example AGCCTAA... format to Arg,Glu,... format.). This concept is what some other technologies call "mediator".

With that in mind we are left with the following links possible:

- $DT_A = DT_B$
    - $O_{ijk}OUT_A\Sigma(data) \neq \varnothing$ and $O_{xyz}IN_B\Sigma(data) \neq \varnothing$.
        * $O_{ijk}OUT_A\Sigma(data) \cap O_{xyz}IN_B\Sigma(data) = O_{ijk}OUT_A\Sigma(data)$ Perfect link [1]

        * $O_{ijk}OUT_A\Sigma(\text{data}) \cap O_{xyz}IN_B\Sigma(\text{data}) \neq O_{ijk}OUT_A\Sigma(\text{data})$ and $|O_{ijk}OUT_A\Sigma(\text{data}) \cap O_{xyz}IN_B\Sigma(\text{data})| \geq 1$
          Possible link [2]
        * $O_{ijk}OUT_A\Sigma(\text{data}) \cap O_{xyz}IN_B\Sigma(\text{data}) = \varnothing$
          - DT_A and DT_B are elemental datatypes. Elemental data link, but it has nothing to do one with the other, for example String person name vs String car license plate [3]
          - DT_A or DT_B is not elemental datatypes. Wrong link and wrong semantics, they just happened to have same name [4]
    - $O_{ijk}OUT_A\Sigma(\text{data}) = \varnothing$ or $O_{xyz}IN_B\Sigma(\text{data}) = \varnothing$.
      Possible link because some names match with each other. In reality we don't have any idea of what's going on but we give a vote of confidence [5]

- $DT_A \neq DT_B$
    - $O_{ijk}OUT_A\Sigma(\text{data}) \neq \varnothing$ and $O_{xyz}IN_B\Sigma(\text{data}) \neq \varnothing$.
        * $O_{ijk}OUT_A\Sigma(\text{data}) \cap O_{xyz}IN_B\Sigma(\text{data}) = O_{ijk}OUT_A\Sigma(\text{data})$
          Possible link, names doesn't match but we have enough semantic information. [6]
        * $O_{ijk}OUT_A\Sigma(\text{data}) \cap O_{xyz}IN_B\Sigma(\text{data}) \neq O_{ijk}OUT_A\Sigma(\text{data})$ and $|O_{ijk}OUT_A\Sigma(\text{data}) \cap O_{xyz}IN_B\Sigma(\text{data})| \geq 1$
          Possible link, names doesn't match and some semantic information fits. [7]
        * $O_{ijk}OUT_A\Sigma(\text{data}) \cap O_{xyz}IN_B\Sigma(\text{data}) = \varnothing$
          Wrong link with wrong semantics and different names. [8]
    - $O_{ijk}OUT_A\Sigma(\text{data}) = \varnothing$ or $O_{xyz}IN_B\Sigma(\text{data}) = \varnothing$. They have different datatypes and we don't have enough semantics to judge in favor, so this will be a wrong link also. [9]

At the end we can group this categories which are the final feedback that the user is going to receive in the program:

- Green - Everything is perfect, only for case [1].
- Blue - Possible link, some semantics works and names are ok, for cases [2], [6] and [7].
- Yellow - We have no idea of what's going on, use at your own risk, for case [5].
- Red - This link is bad, for cases [3], [4], [8] and [9].

9.10 Semantic correlation between operations

Here I'm going to talk about a concept that is very similar to linking types, but linking operations instead. As explained before an operation can be semantically linked with a data with the relationship of "has_input" and "has_output". Essentially if an operation A has output X, and another operation B has input X, they are semantically correlated.

We will explore now the different combinations as we did with the types links. Luckily this time, format is irrelevant. Two operations can make sense to link or not, but we do not care about going down to data structure levels to check if they are actually compatible.

One more thing. The same concept of data being a specialization of another data continue to apply in here. If an operation has output the specialization of another data from another input of another operation, the link is still valid. But can't say the same if a generalization is linked with a specialization. When we say that operation A correlate with operation B we means the outputs of A correlate with the inputs of B. Note that if A correlate with B is not necessarily true that B correlate with A.

- $O_{ijk}\Sigma$(data|output) $\neq \varnothing$ and $O_{xyz}\Sigma$(data|input) $\neq \varnothing$
  - $O_{ijk}\Sigma$(data|output) $\cap$ $O_{xyz}\Sigma$(data|input) = $O_{ijk}\Sigma$(data|output)
    The semantic data match and the correlation is perfect. [1]
  - $O_{ijk}\Sigma$(data|output) $\cap$ $O_{xyz}\Sigma$(data|input) $\neq$ $O_{ijk}\Sigma$(data|output) and $|O_{ijk}\Sigma(\text{data|output}) \cap O_{xyz}\Sigma(\text{data|input})| \geq 1$
    Some of the outputs can go in the inputs. This still makes sense to link although probably you will need another operation to link with the input. [2]
  - $O_{ijk}\Sigma$(data|output) $\cap$ $O_{xyz}\Sigma$(data|input) = $\varnothing$
    Semantic doesn't match at all, these operations have nothing to do with each other or the semantic annotation is incomplete. [3]
- $O_{ijk}\Sigma$data|output) $\neq \varnothing$ or $O_{xyz}\Sigma$(data|input) $\neq \varnothing$

  The semantic annotation is incomplete. We can't judge in one way or the other. [4]

This will translate into the following color scheme

- Green - Everything is perfect [1].
- Blue - Possible link, some semantics works [2].
- Yellow - We have no idea of what's going on [4].
- Red - This link is bad [3].

9.11 Semantic interchangeable

If we have operations A, B and C, and A semantically correlate with B, and B correlate C, and A correlate with D, and D correlated with C; then we will say that B and D are semantically interchangeable.

9.12 The puzzle view

Until now we have talk about sets and properties in a mathematical way. We can simplify everything if we consider the two properties of link and semantic correlation as pieces of a puzzle.

Two pieces of a puzzle can be snapped together if the shape match. That is what we call linking. The puzzle makes sense if the final picture looks like the one in the cover of the box. That's what we call semantic correlation. We can make a puzzle out of pieces of several puzzles, we make sure that the pieces can snap with each other and at the end we will have an aberration. That's is what happen when we link things and we don't care if they semantically correlated. Alternative, we can make a pretty picture if we collect different pieces which images match the box's cover, but if the pieces won't snap with each other the puzzle will collapse. This is what happen if you don't link properly.

Following that principle let's take a look at the following diagrams.

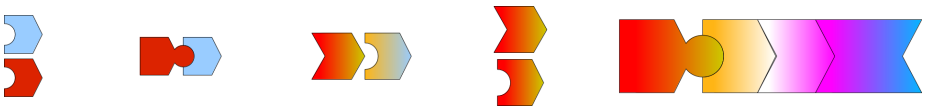

Figure 10: The puzzle view

From left to right: two operations that can interchange links, two operations that can be link, two operations that semantically correlate, two operations that are semantically interchangeable, a chain of operations which are linkable and semantically correlated one after the other.

In total we have 6 different properties:

- A can link B
- B can link A
- A can interchange links with B
- A correlated B
- B correlated A
- A can interchange semantic with B

We can take a look at all possible combinations and see if we can find any interesting properties:

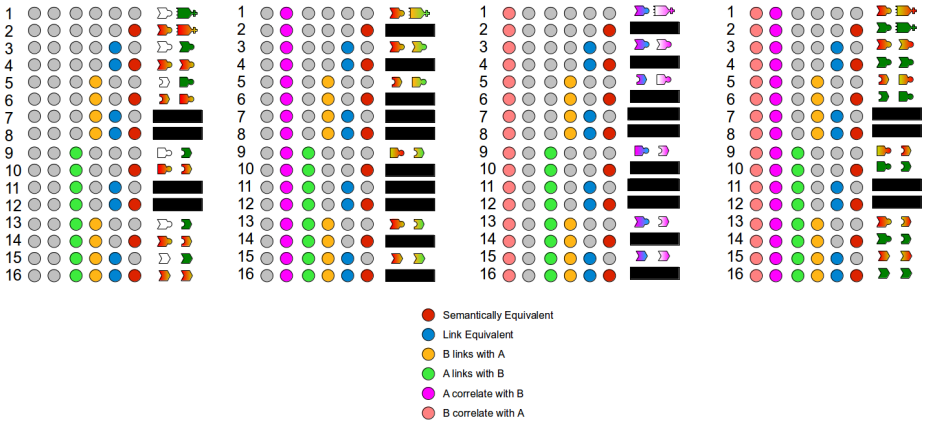

Figure 11: A true table with all possible combinations of properties

Here are listed the 64 possibilities that we have for linking two operations. The figure is divided in 4 groups of 16 possibilities each. The first column represent when the service B correlate with A. The second column when A correlate with B. The third column when A can link with B. The fourth column is when B can link with A. The fifth when links of A and B are equivalent. The sixth when A and B are semantically equivalent. Next to each row a graphical representation of the puzzle view for that case. Some cases are impossible to reproduce, they are label with a black tag over the puzzle.

Following up, this properties arise:

- If A correlated with B and A links with B then the two operation can be set together.
- If A and B are both semantically equivalent and link equivalent, then A and B can swap each other. This means that they are equal for our purposes.

This properties are important when it comes to WS discoveries.

9.13 The Workflow level

We still don't know if it is a good idea to place A after B. We can make sure that the outputs of A are semantically compatible with the inputs of B. But that doesn't mean that the result makes any sense.

For example, the derivative of a function take a function and gives a function. We can change that WS with an integration operation. The inputs and outputs are still the same, a function, but the result is completely different, an integral rather than a derivative. That is something we can prevent with the semantically related part. However, we can link the derivative with the integration operation and the result will be the first input again. That respects the links rules and the semantics rules. However that would be a silly workflow because we will get the original input again and accomplish nothing.

Now that we have a chain of WS that makes sense as links and sense as semantic correlation. How do we evaluate it as a good workflow?.

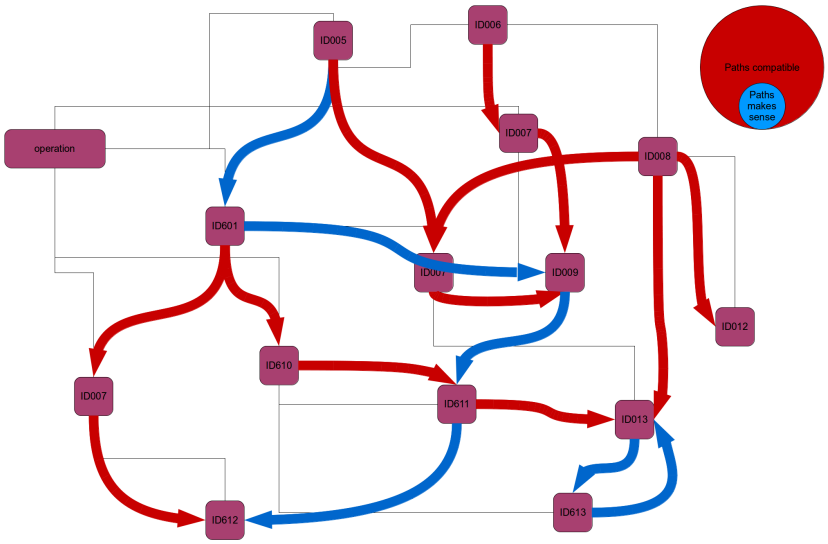

Figure 12: A representation of every possible workflow

A representation of every possible workflow in red, compare with every workflow that actually makes sense in blue. We still haven't done anything to be able to tell if we are in a red path or a blue path. But if we follow the rules set before we are at least in a red path for sure.

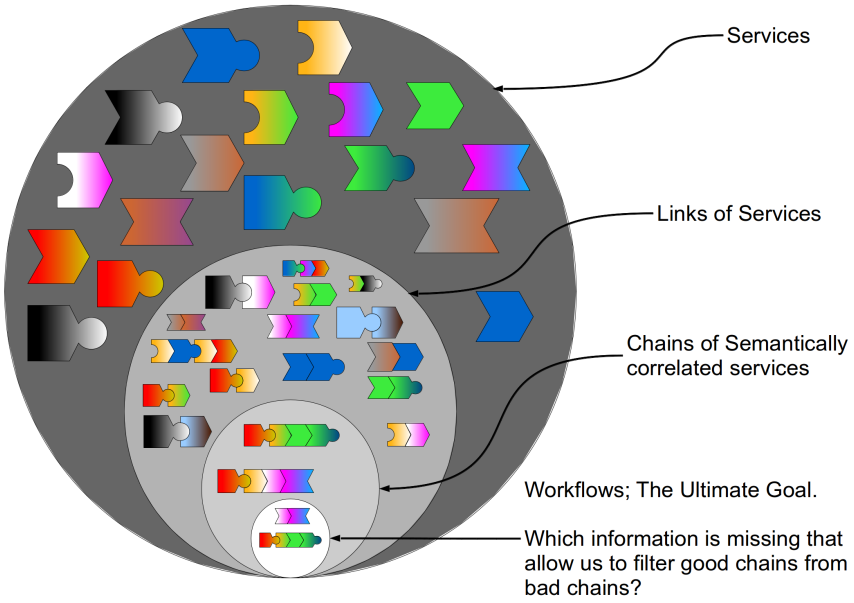

Figure 13: An overview of the different levels of information.

We start with simple services scattered around and we start joining them if they can be linked. Among those which are link we select only those who has a nice semantic correlation. Now we need to filter those to reach the ultimate workflow level.

I personally think that if the service is properly annotated then we can distinguish between good and bad workflows using the Topic sub-ontology.

9.14 The Workflow score

The semantic structure of a graph, and the nature of a workflow also as a graph, gives us the possibility of devise the concept of Workflow Score.The workflow score represents how good or bad a workflow is with a numerical value. It goes from 0 to ∞ , being 0 the most desirable value possible. It measure the distances between topics from different operations. If you ascend from topic to topic in the ontology, the penalty is almost nothing. However if you change from one branch to another you get full penalty.

Lets suppose we have an operation Z, which have two kind of semantics annotations $O_z\Sigma$(topics) and $O_z\Sigma$(operations). Each of the $O_z\Sigma$(operations) can have several topics associated with it through the relationship "has_topic" from EDAM. We extract those topics from there and we added to the set of topics of $O_z\Sigma$(topics); at the end we have a set of unique topics elements from EDAM for operation Z, and let say that the total of topics is |Z|.

We can have several operations that connect with Z. Each one of these operations, O_{x_1}, O_{x_2}, ... , O_{x_n} also have a set of topics defined just as before which have a total of $|x_1|$, $|x_2|$, ... , $|x_n|$ each.

For each one of the operations that connect with Z we can create a matrix of $|Z| \times |x_i|$. The element $A_{a,b}$ of the matrix is the distance from topic a in the Z set and topic b in the x_i. The peculiarity of this distance is that the total distance will be the 10% of the distance if we can go from topic a to topic b without changing branch in the EDAM graph for topics. Otherwise it will just be how many jumps we need to take using the 'is_a' relationship. In this way we rewards operations that are closely related.

Once the matrix is complete we have exactly n numbers for operation Z. Each of them comparing the distance for each of the operations. The total score for operation Z will be the sum of all those numbers divided by n. We take the average of the operations because we don't want to penalize operations with many inputs.

Now; each operation of the workflow have it own score calculated as we just described. A workflow is an acyclic graph made of operations as nodes. The Workflow Score for a given part of a workflow would be the sum of all the previous nodes scores.

With this variable now we can take measures for many valid workflows. We define valid workflow when all links between elements or operations are green or blue. This measure can be represented graphically in a scatter-plot where the X-axis would be the size of the workflow (greater distance between nodes) and the workflow score. With such scatter-plot we can also made a regression model and find the best equation that fits the sample we took from empirical data. We already have a vast database of workflows which are valid and rated by users in myExperiment; so mining the information from there, apply the Workflow Score to them, and add the result to the scatter-plot would be an easy task.

This function will determine a region in the 2D space where the workflow score is suppose to be. If the points are more scattered this region will be wider and viceversa. For us to declare a workflow valid it should be around that region. In a more formal definition, we can define a distribution with a population average and standard deviation for each of the possible dimensions of a workflow based on the sample data we gather. For a given worflow with its own workflow size and workflow score, we can now determine what is the chance of that being a valid workflow according to our distribution. We can give a 95% confident interval center in the average of that distribution and if the workflow score fall outside that interval we discard the workflow as non-valid.

The ultimate goal of this is that we now have a method to, by brute force, explore all the possible workflows between two operations.

Suppose we know from where to start and which operations is the final one. We can even define the desired output of such operation. We can start a branch and bound algorithm to create every possible workflow in between that leads to the desired output (if any). If one branch goes higher or lower that our interval threshold for that given workflow size we stop exploring that path. Notice that we are talking about valid workflows only, so the possibilities are quite narrow.

In this way we can get the computer to test experiments for us automatically until we find a series of transformation for a desired output, much more quicker

than a human doing it manually.

9.15 Summary

So far we discussed the mathematical model of a WSDL and made some predictions on properties which are yet to be tested. We saw also that current technologies are not good enough for the idea presented here. What comes now is a tutorial on how to use the program to test this ideas.

10 The WSDL-Workshop program

The program was developed following the theoretical analysis mentioned in chapter 9 in page 30, This section will explain you how to use the program WSDL-Workshop in detail.

10.1 Design choices

10.1.1 GUI

For the design of the Graphical User Interface (GUI) I used a concept called Experiential Metaphor [40]. This mean to try to evoke in the user a feeling or a visualization of abstract ideas. In this case the workflow can be compared to a jiggsaw puzzle just like we saw in chapter 9.12 in page 42. Hence the use of pieces to talk about operations. These kind of puzzle however rely on a unique 2D geometry for each of the pieces. To speed up workflow building we use the guide lights to give the user a preview of what is going to happen if you connect two datatypes before they get connected. We even get the user the filter to be able to find a piece that fit another one automatically, something that will render a jiggsaw into a very easy challenge. Likewise we want the user to be able to construct the puzzle for his problem as quickly and easy as possible, even if that means spoiling the fun of having to figure the answer out all by himself.

At the same time we tried to give each element unique information through color coding. All operations are orange, all input/output are either green or purple depending if they are simple or complex. Links goes from red (bad) to green (good). When something is selected or can be interacted it highlight automatically, as in all elements in HTML that exploit the CSS effects. Aesthetically can have some improvements with the choice of the geometries, font styles, size of the elements, animations, lights effect and so on. But the general idea that colors convey information still remain.

10.1.2 EDAM optimization

In EDAM you have relationships to traverse the graph from the bottom to the root but not the other way around. I implemented the data structure so you can transverse it in all directions. You also won't find IDs in the node of the graphs but pointers to the actual elements. Please refer to the source code where you can find much more detailed information about algorithm used to traverse graphs.

10.1.3 Drawing process optimization

The program is designed so it only draw sections which have changed. For example if you drag one element to another place the program doesn't recalculate coordinates in the drawing function; the dragging action order the element to self-validate and the drawing function later on get the data. Beside minimizing the resources required for drawing it also helps for a multi threading render of the screen. This is not necessary because the program doesn't use complex 3D graphics or lightning scenes, but it is nice to have in case somebody decided that it will work better in another environment.

10.1.4 HTML functionality vs JavaScript functionality and Server workload

There was a great deal of effort invested into letting the browser adjust the window using only HTML+CSS coding and minimize the use of JavaScript as much as possible. Also the server only works when sending the WSDL and ontology files and when you ask to save your data into a file. It was also an aim to make the application as lightweight for the server as possible to reduce the cost of deployment.

10.1.5 Documentation

The source code also offer a lot of comments to guide other developers about design choices and help them to make modifications. There are also standard UML diagrams available. If you want to implement this program in another language you shouldn't find any troubles other than making the translation; no redesign should be required. Furthermore, everything was done with standard JavaScript libraries and no third party support is necessary.

10.2 License

I decided to license it under a Creative Common Share alike - Non profit - By license. [41]

Here is a list of works or tools that I made use of:

The EDAM Ontology was developed in the University of Bergen. The WSDLs and XSDs which accompany the source code were taken from the University of Bergen also. Please refer to this institution if you want to use any of that work.

During the process of this work I made use of Geany [42] editor in a Ubuntu 10 [43] most of the time. And Netbeans [44] with Glassfish [45] server plugging for certain demonstration purposes running in a Ubuntu 12. The private server in which I'm running this program is a Linux Mint Server Edition [46] which is based in Apache. All of this have a GNU license [47]. The icons in the toolbar and in the side menu have a LGPLv3 [48] license. The thesis written document was made with Kile [49], a LaTeX editor, with license GNU v2.

10.3 Prerequisites

10.3.1 Chromium Browser

Chromium [50] is a open source web browser. It has an 100/100 Acid3 score and is free. It is licensed under several GNU-like licenses. The commercial version derived from this browser is call Chrome. The browsers are 99% similar; the main difference is that the first is open source while the latest is not.

10.3.2 Computer Minimal Specs

Tests show that the program and the server runs on a 1GHz Intel/512MB RAM computer without any problem. The program itself is fairly lightweight and it

will only run slow at the very beginning, while it initializes the ontology and parses the WSDLs. This process shouldn't take longer that 5 seconds using the default WSDLs,XSDs, and ontology files.

You should be able to run it also in any device that has a browser with JavaScript enabled, such a smart phone or a tablet; and given that the browser you are using has a compatible rendering engine with WebKit [51], such as Safari [52] or Chrome, for example. The program will scale automatically to the size of your window, so if the screen is very tiny, as long as you have slim fingers or a pointer device, you should expect no troubles.

10.4 Starting the program

In order to start running the program you must do three things:

A) Setup a server. As stated earlier, the most simple solution is to run Netbeans and start the Glassfish server from there. Notice that if you want to be able to save or load files from or to the client side you have to give POST permission to your server. But this is not necessary to run every other functionality; since the server has access to a default directory of WSDLs it is not essential to modify this.

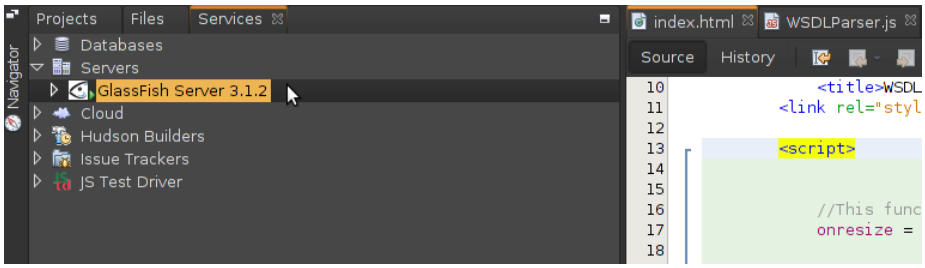

Figure 14: An example of the server Glashfish running.

Notice the green "play" icon that indicates the server is running. In the image we also see other options that Netbeans offers and part of the window with the source code to the right.

B) Get the source code into place. If you use the Glassfish option add the docroot folder directly into the docroot folder of the Glassfish directory, and it should run fine. You can download it from `https://github.com/rafanozal/WSDL-Workshop`.

C) With a Chromium Browser, go to the index.html page. If you run your server locally, the default address is http://localhost:8080/index.html.

Once everything is running the program will start automatically as soon as you visit the index.html. The program will first show you a welcome splash screen where you can read "WSDL Workshop". In the background, the program will start parsing the default WSLDs and XSDs which come in the /wsdl and /xsd folder. It will also parse the ontology in /res/ontology, and create the

necessary data structures in memory. Once finish it is ready to use; the splash screen will turn white as soon as you click anywhere.

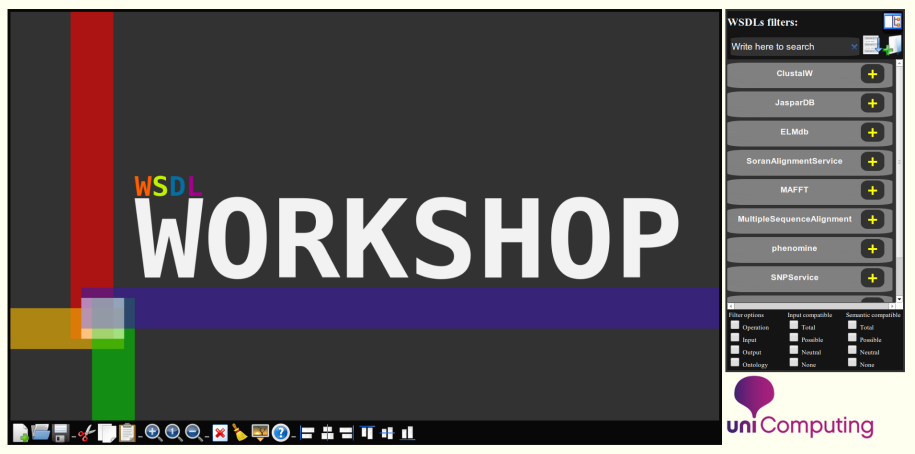

Figure 15: The program is running in the client side and is ready to use.

10.5 Sidebar; WSDL listing and discovering functionality

In the sidebar you will find 3 main sections. The WSDL filter on top, the WSDL entities in the middle, and some extra filtering options in the bottom.

10.5.1 Top

In the top you can read "WSDLs filters:"

Tree/List view


To the right of the title you can find the tree/list view swapper. What this does is change the middle view. By default it is set to start in list mode. So you will see all the default WSDLs in the middle. These WSDLs start with no filter applied to them. If you press this button it will change into a tree view where you can find the ontology viewer. This viewer will allow you to discover WSDLs related to a topic that you search. You can tune your filtering with the options in the bottom of the sidebar.

Filter text

This bar allow you to write some text. As you write you will find a collection of operations, inputs, outputs and topics that are displayed to the left of it. These are all the concepts which are currently available in all the WSDLs in the program. These concepts contain the text that you inputted in their title. You can modify this filter with the options at the bottom.

Sort by name


This option will sort the WSDL list alphabetically, or in reverse if it

was currently starting by A. If the view is set in Tree-mode, it will sort the tree accordingly.

Add new WSDL

 When this icon is clicked it will start an explorer window so you can localize your own WSDL file and import it into the program. If the program is not able to recognize and parse the WSDL you will be prompted with a comprehensive error message. Otherwise the WSDL will be added and be ready to use. Make sure that your filter options are compatible with the WSDL you just enter, otherwise it won't be shown until you set the filter so that the WSDL fulfills all the requirements.

10.5.2 Middle

In the middle of the side bar you can encounter two types of views: the list view and the tree view. By default you have the list view and you can change this with the button at the top of the sidebar.

List view

In the list view you can find the WSDLs listed in the order they were parsed. You can sort them alphabetically by their title if you wish so. Each WSDL can be expanded with the "+" button on its side. Once expanded you will see the different ports that compose the WSDL. Normally you will only find one port per WSDL. Ports can be expanded too; once done, you will see the operations list that compose each port.

The operations are the entities that you can add to the canvas. You will find two buttons in each operation. You can expand the operation to see a preview of the inputs and outputs which will show their names and datatypes. The other thing you can do is to add the operation into the workflow canvas.

You can collapse the operation, port and WSDL by clicking in the "-" sign which will replace the "+" sign once you expand them.

Tree view

In the tree view you can explore the operations of the WSDLs you have available from the ontology point of view. For each ontology entry you will find also a "+" sign which will expend the concept, or a "-" sign which will collapse it again.

You can find four sub-ontologies here. Operations, Data, Topic and Format. Each of these concepts, once expanded, will show the concepts that hangs under them. Please refer to the relationship "is_a" in the ontology section in chapter 8.5.2 in page 26 to know more.

In the Operations ontology you can find operations which have the given concept annotated in them, and you can add them the same way you do in the list view.

In the Data ontology you can find operations which have either an input or an output annotated with the concept which you are exploring at that moment.

In the Topic ontology you can find operations which have the current topic annotated inside. This could be at any level, the operation, one of the data, the element of the data, the type of the data, and so on.

In the Format ontology you can find operations which have either an input or an output annotated with that format.

In general this view is useful for exploring the ontology and discover operations related to a concept that you didn't think it could exist or didn't remember.

10.5.3 Bottom

Here you can find filtering options that will hide or show WSDLs or operations in the middle section of the sidebar.

Filter options

On the top of the sidebar you can find a text searcher. As you type in that bar you will find concepts that pop out. By default you will find this filter set to nothing, and it will look for that concepts in the operation names, inputs names, outputs names, and semantically annotated topics.

You can however change the filter option. If you select one ore more, it will filter by those groups, and will show you operations that match all of the cases. For example if you select "operation" and "input" and enter the word "sequence" it will show you WSDLs which contains operations with the string sequence and at least one input with the name sequence.

You can combine this with the tree view as well.

Input compatible

If you have selected an operation output in the canvas you can use this option.

This will filter the operations in the middle of the sidebar to only those which are compatible (or not, depending of your selection) with the current selected output. So for example if you want to make sure to find an operation that can receive the string that your current operation is outputting and you have selected, you will make use of this option, and select the options "yes" and "possible". Please refer to the chapter Making Links Simplified in section 9.9 in page 39 for a complete documentation of what each of them exactly means.

Semantic compatible

This works similar to the "Input Compatible" option. When you select an

output of an operation or a single operation, you can filter the WSDLs to show only WSDLs which operations are compatible (or not, depending of your selection) with the selected piece in the canvas. Please refer to the chapter Semantic Correlation in section 9.10 in page 40 for a complete documentation of what means each of them exactly.

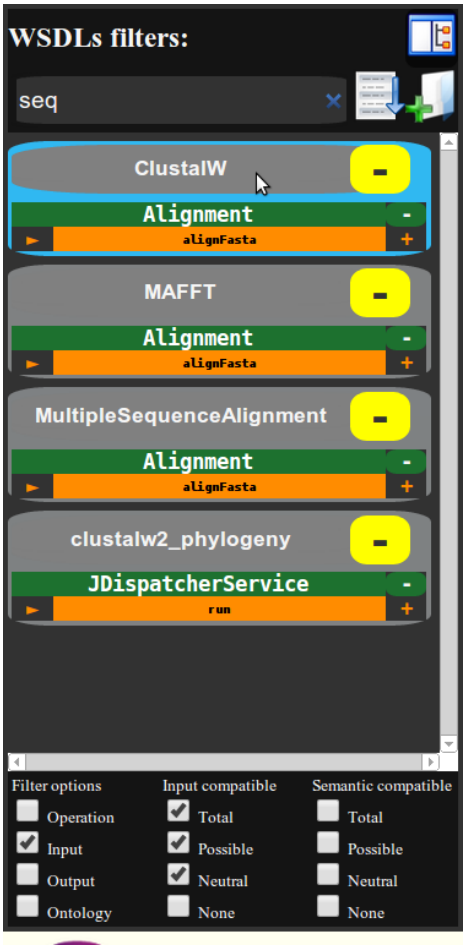

Figure 16: An overview of the entire sidebar.

The user has decided to enter the text "seq" to look for operations that have at least one input call "seq", probably referring to the word "sequence". At the middle, four WSDLs match the criteria. Each one of them show only those operations that match also the criteria. In this case only one operation in each does.

10.6 Tool bar; creating a design environment

This is the bar that appears in the bottom of the screen and below the canvas section. This is a series of tools to interact with the canvas.

New

 Reboot the canvas and start a new workflow project.

Open

 Open an old workflow project file. Your current project will be lost if you don't save it first.

Save

 Save the current project into a file in your device.

Cut

 Delete the selected operation including links and save them in a temporary variable. If you cut something else your old operation will be lost. If you copy something else your old operation will be lost.

Copy

 Copy the selected operations including links and save them in a temporary variable. If you cut something else that will replace the temporary variable, if you copy something else that will also replace it.

Paste

 Put the temporary variable inside the canvas. If there is no temporary variable nothing will happen.

Zoom in

 This will increase the zoom in the canvas and everything will appear bigger. You can also do this with the keyboard shortcut "+". The program limit the zoom level to +5; which is a screen 32x smaller than the original.

Zoom 1:1

 This will modify the zoom to return to the default state.

Zoom out

 This will decrease the zoom in the canvas and everything will appears smaller. You can also do this with the keyboard shortcut "-". The program limit the zoom level to -2; which is a screen 8x bigger than the original.

Delete piece

 Delete the selected operation from the canvas. You can also do this with the button in the operation itself and with the keyboard shortcut "Del".

Delete all

 Delete every piece in the canvas. In practice this has no difference with the New workflow option except that you can still undo this.

Take picture

 Save the current status of the canvas into a PNG image. The picture will be opened in a new window.

Help

 It will display the manual in a new window.

Aligning options

 Its allow you to move the operations so they align nicely to each other. You can find the six standard aligning option that you will find in any other design program. You need to select two or more pieces to see the effect.

10.7 Piece; the operation representation

A piece is each of the operations that you can add to the workflow. The piece have several elements. On top of it you will see a black bar with several buttons. Then you will find an empty space with the operation name. Then a bigger space with the inputs and outputs. And finally the foot bar with the final options. In figure 18 in page 58 you can see a complete representation of a piece.

10.7.1 Top bar

This is the thin black bar and the orange thick bar on top of each operations. You will find a series of options in it.

Delete operation
You can delete the operation with the red button. All links coming from or coming out the operation will also be deleted.

Minimize operation
This is the yellow button. It will hide the middle space. You can still see links that come out of the piece or into it. You can undo the process by simply clicking in the button again. If two operations which are connected are minimized then only the operation link will be show, and the links in between inputs and outputs of those two operations will be hidden.

Expand semantic view
This button will show you all annotations which are related with this piece. The operation annotations are shown at the button of the piece, while the data annotations are displayed below each data. Sometimes the data annotations can be a bit overwhelming. In that case, if you collapse the operation box it will only show the annotations for that level and nothing from the levels bellow.

Title
In this section you will find the title of the operation. At both sides, you can find the operation links if the operation is connected with anything.

Semantic Correlation Link
When you link types between operations the operations headers will be automatically linked. The color scheme of the operation link represent the semantic correlation between then. Please refer to the Semantic Correlation chapter in section 9.10 in page 40 for more information.

10.7.2 Middle space

This is the space where all input and output are collected.

Input and output connection boxes

In order to make a link between two operations just click in an output first and then in the input you want to link. You will see a preview of the link next to the input in the form of a highlight color which indicates what kind of link it will be. Please refer to the chapter "Making links simplified" in section 9.9 in page 39 for a complete documentation of what each of them exactly means.

Once you made a linked between an output and an input, the operations will also be link. Remember that the quality of the operation link has nothing to do with the quality of the links of it's outputs or inputs.

Note that there is an important restriction when making links. You shall never form a cycle. You can however copy the same operation several time and link one after each other.

Figure 17: Two operations about to be linked

In this example, the output resultFasta is highlighted because it has been selected. The operation "run" shows the preview of each input and the semantic correlation of the operation itself with "getResultFasta". In this case, only the input "sequence" is evaluated as a good idea. The rest of the inputs have not the same datatype (in red), or haven't being semantically annotated (in yellow)

10.7.3 Foot bar

This is the last part of the piece at the bottom of it.

WSDL Name

This is a reminder of from which WSDL did the operation came. Some times the operations have the same generic title; as in "getDataByID". So it is nice to distinguish one from the other by the original WSDL file.

Color roulette

This gives user organize operations by background color.

In order to change the color simply click the rectangle, and without release the click, move the lever around the color wheel to select a new one.

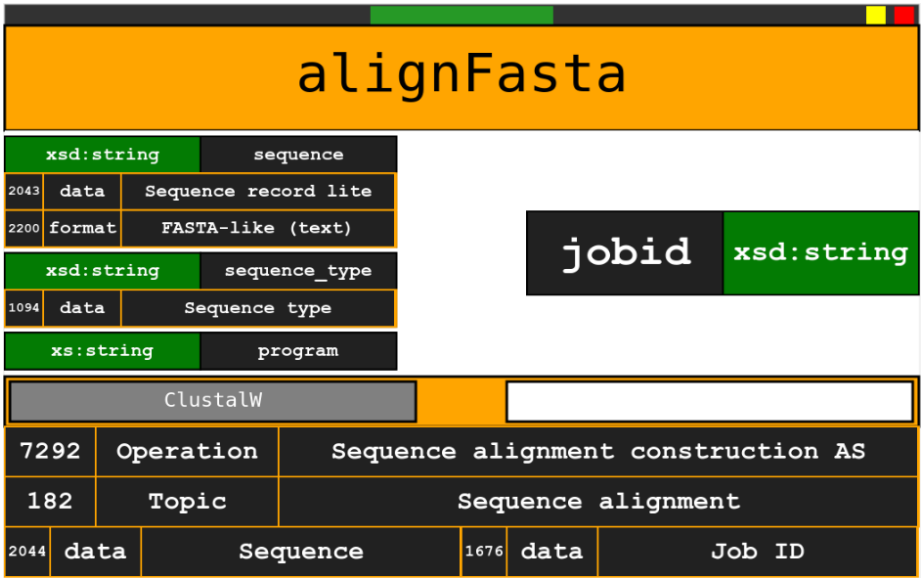

Figure 18: An example of a piece showing it semantic annotations.

Notice that neither program not jobid have being annotated and thus nothing is displayed under them. Bellow the foot bar the semantics for the operation itself is shown. This includes the relationships of has_input and has_output of the operation semantics, to the left and to the right respectively.

10.8 Canvas; the drawing board where everything work together

This is where you will interact with the program. Here is where the pieces are added and where you have to connect pieces witch each other. If you click in the adding button of the sidebar a new piece will appear in the canvas.

10.8.1 Scale anchor

When an operation is clicked you have the opportunity of change the size of the piece. The links and texts which are inside will automatically change size too, and keep the same proportion of the original box.

10.8.2 Multiple selection

If you click on an empty space and drag the mouse you will form a rectangle. Once you release the bottom, the program will select every piece which is inside the rectangle. Multiple pieces can be deleted at once or be move.

10.8.3 Deleting Links

If you want to delete a link or more than one link, click with the right button of the mouse and drag a line. Every link that the line cross will be cut and deleted. If this leave two operations unconnected, the operation link will be sever as well. If you try to delete an operation link nothing will happen. You must cut the link between two operation inputs/output for successfully cut the link at the operation level.

10.8.4 Workflow score

If you click the middle mouse the canvas will shift into workflow view. Here you can see the workflow score for each operation and the accumulated score for every operation that is connected before.

In each operation you will see the topics which have been annotated. Also you can see a list of operations which is connected to it. If an operation have no previous operations or no topic the workflow score will be 0, and a message regarding this will be show.

Please refer to the chapter of Workflow Score in section 9.14 in page 45 for a more detailed insight on how does it work and what does it means.

10.9 Keyboard shortcuts

Arrow keys
: Pressing the keys have two different effects. If you have selected one or more pieces it will move the pieces in that direction. If you haven't select anything it will move the camera in that direction.

  If SHIFT is pressed it will move the camera or the pieces ten times faster.

+/- keys
: These keys will increase the zoom or decrease the zoom.

Delete key
: If one piece or more are selected it will delete them.

Ctrl + N
: This will make a new workflow.

Ctrl + O
: It will open the dialog for opening a workflow file.

Ctrl + S
: It will open the dialog for saving a workflow file.

Ctrl + X
This will cut the selected pieces.

Ctrl + C
This will copy the selected pieces.

Ctrl + V
This will paste the selected pieces.

10.10 An simple test case

In this section we will see an example of simple workflow example. Let fetch sequences from a database, then align them, and finally lets construct the phylogenetic tree of the collection of sequences give it to us by the alignment. This is represented in figure 19

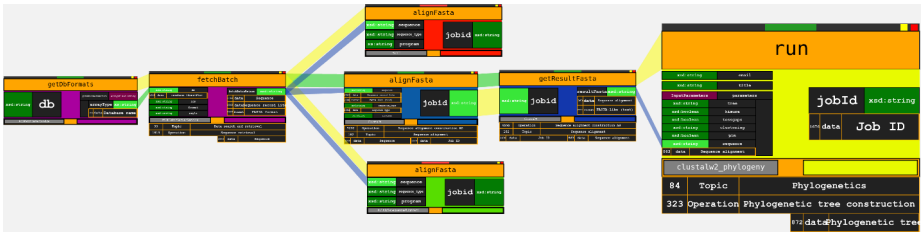

Figure 19: An overview of a workflow.

The workflow is branching at the sequence alignments operation and show that only the operation that belongs to the WSDL ClustalW has been annotated. Following that branch the workflow continue into a phylogenetic tree construction operation. The user decided to change the background of each piece, and assign a color to each WSDL. So pieces with the same background belongs to the same WSDL.

Navigating the sidebar you can find a WSDL called 'WSDBFetchServerService'. We should add two operations from there, 'getDbFormats' and 'fetchBatch'. The element 'getDbFormatReturn' from the first operation is an array of strings and each of these strings are annotated with 'Database name'. We can connect the string to the element 'db' of the second operation which is annotated with 'Database identifier'. The program creates a green link because 'Database name' is actually an specialization of 'Database Identifier' so both elements are perfectly compatible. At operation level we can't say anything because 'getDbFormats' has no annotations at the operation level. Even though we know that this two operations make sense link, the WSDL is not sufficiently annotated and the program can't judge in one way or the other.

Now lets add two new operations from a WSDL called 'ClustalW'. Operations 'alignFasta' and 'getResultFasta' can be found inside. If we link the first to the second we can see the opposite case from before. The element jobid, in both cases, has no semantic annotations so we can't judge that link as valid or invalid; hence the yellow link is created. However at the operation level we see

that they are actually the asynchronous version of the same operation, so they are awarded with a green link.

If we select the output from 'fetchBatch' called 'fetchBatchReturn' we will see that the program recommends to join it with the input 'sequence' of alignFasta. In this case is possible to do so at datatype level because both are annotated as sequences (plus they have similar format). Also, they can be link at operation level because 'fetchBatch' is a 'Sequence retrieval' operation which will give you a sequence, and 'alignFasta' is a 'Sequence alignment construction' operation which receive a Sequence. So they are perfect for each other as the program recommended.

Lets make some use of the filter. If we select 'fetchBatchReturn' and we look for 'sequence' that is either fully compatible or possible we will find two more possible operations. Both of them are call 'alignFasta', one from 'MultipleSequenceAlignment' WSDL and the other one from 'MAFFT' WSDL. If we try to link 'fetchBatch' with these we can see that the datatype is compatible, but at operation level we don't have any information because neither of them has been annotated at operation level; so we will continue with our original branch.

Finally, lets add the operation 'run' from the WSDL 'clustalw2_phylogeny'. We select the output 'resultFasta' from 'getResultFasta'. Among the many inputs options that the operation 'run' have, the program only recommend to link it with the one call 'sequence'. At operation level we don't have information despite the fact that the 'run' operation is semantically annotated with the operation 'Phylogenetic tree construction'. This is because the EDAM ontology doesn't provide a relationship of 'has_input' to that operation. If the EDAM is updated to have this information then the program also recommend the link at operation level.

10.11 Summary

In this chapter we saw the design choices and how to start the program and run it at different levels. With all the information given so far the user should be able to understand every detail of the program and the theory behind it.

11 Discussion

WSDL-Workshop tests shows that the developed theory of how to use semantic annotations to differentiate good from bad workflows is very promising; for instance the test case gives you the correct information wherever there are semantic annotations, and the filters works perfectly for discovering relevant operations for our interest. This section is dedicated to ask possible questions about design choices and for me to voice some detailed opinions.

11.1 EDAM

The user is responsible for introducing a properly semantically annotated WSDL in the program. The discovery of WSDL is a huge problem on its own as we saw in the introduction, but it is even more difficult is to find WSDLs semantically annotated; and to be precise with the EDAM ontology in particular. There is a discoverer for EDAM SAWSDLs which can be found in EMBOSS [53], but currently the options are pretty limited to what you can find at the Computational Biology Unit (CBU) of the University of Bergen (`http://cbu.bioinfo.no/wsdl`), what you can annotate in eSysbio (`http://esysbio.org/about`) and some extra custom SAWSDLs made for testing the this very application. All of them are already added by default in the sidebar of the application. It would be very easy to add a repository from where the program fetched EDAM SAWSDLs automatically and keep it updated, but at the moment such repository doesn't exist. Since the collection of SAWSDLs is limited, the concept of Workflow Score couldn't be tested properly, and it cannot be said that it works or doesn't work for finding the best chain of operations automatically.

The EDAM ontology have a couple of flaws that affect the behavior of the program. EDAM doesn't have any entries for asynchronous operations. These are the kind of operations that you submit your inputs and gives you back a job identifier. Later on, when your query is finished, you can submit your job ID and retrieve your actual outputs. This can be easily corrected if each operation of EDAM which is a leaf in the acyclic graph, get the extra specializations of two operations that represent the same operation, but one have as output the job ID, and the other one have as input the same job ID. This has being successfully tested in the program, and if you look in the ontology file you will find custom entries with the text "created_in: 'WStest"' which are the custom operations added. A more efficient way of doing this would be to add a keyword to the annotation system, so you don't need to modify each operation in the ontology. That keyword could be used as for example "sawsdl:http://edamontology.com/operation_1234:AsynchronousOutput" to refer to the element Operation 1234 in the ontology, and to express that this particular WSDL will give you back a job ID instead of the normal output.

The second problem with EDAM is that the program allow you to link a job ID from WSDL X to a job ID from WSDL Y. This of course won't work, it will try to fetch the results of a job in WSDL Y that nobody has submitted yet. It can be fixed by either annotating the service with a custom JobID semantic which shall be a specialization of the ontology entry Data 1676

"Job ID". The problem is that you will have to add an entry for almost every asynchronous operation out there, which would be mathematically correct but quite overwhelming. An easiest way would be to make an exception for Data 1676 in the rules for making a link. And if you try to link something with that output, it must be with another Data 1676 which belong to the same WSDL.

11.2 Choice of language

One of the aims of this thesis is to provide the user with an easy access application which doesn't need to install special packages or tinker with the operative system. The two options here are either make an online application or make a desktop stand alone application. We decided to go for the web application. For this case there are four main approached. Either you make it with Flash, Java, HTML5 or you run it in the server side.

If we provide an application for the user we want to have the lower monetary cost as possible for us. It is possible to be done in the server-side and use any other language code but in that case you will be putting stress on the server which will increase the cost for the server runner; plus this is likely to decrease the experience for users because bottlenecks, slow internet connections and many other reasons. So this option was out of the equation.

Flash is proprietary, non-standard, breaks conventions associated with normal HTML pages, has low performance, has withdrawn support to mobile devices in favor of HTML5, and the drivers for 64-bits version of Linux have been neglected in the past. Because all of these Flash is definitely not an option.

I chose HTML5 because of the simplicity for the user. You just need a web browser with Internet access and you are ready to start using the application. Java is the next option, but because of security concerns [54] [55] [56] [57] and similar or worse in efficiency [58] [59] I discarded that option. Recently, even the major browsers stopped supporting and enabling Java by default [60].

It is impossible to make this program only in HTML + CSS. There are still some issues with JavaSctipt that we need to review. Some users may have JavaScript blocked in their browser for privacy or security reasons. Many developers make an abuse of its functionality in order to flood the user with advertisement, take private navigation data or other nefarious ends. Normally is considered a bad programming when you cannot navigate a website without using JavaScript. For example trying to see a slide share of pictures and forcefully has to use JavaScript is a big mistake. That is because you can provide the user with the HTML links to the images instead of forcing the user to use your interface; as a thumb rule if you could have done it in 1995, then JavaScript just add pretty effects and no functionality. In this case JavaScript is justify because it doesn't just increase the functionality of the web or make it pretty or something mundane; it is needed from the computer to find data for me and make complex calculations such as the Workflow Score.

JQuery is a JavaScript library [61] developed to simplify the client-side scripting. When making an application in JavaScript one need to think about

using JQuery because sometimes can make things easier. Among the bad things is that you depend of third-party developers that maintain JQuery and it is quite difficult to read code in JQuery. The good things is that it makes easier to do "pretty" things and the algorithms are quite efficient. I decided to not use JQuery in the thesis. Another aim of the thesis was that you should be able to pick up the program, understand was going on inside the project easily, and modify to your will. However; in a commercial and final version of this program I would consider the trade off of increasing efficiency that JQuery provide.

JavaScript however is very difficult to program. Not because the syntax is complicated but because it is prototype-oriented. This means that, for example, if a variable doesn't exist, it won't complain about it and won't give an error until you reach the part of the code where that variable is used. Things like this increase the testing time of an application exponentially and increase greatly the total developing time. Based on that, I will now considered do this for a desktop application instead. But still, making the browser application make the application more available and easy to use. Since this is an experimental application to test some hypothesis I will do it again with JavaScript. But if a commercial application is needed, the choice of a desktop application would be quite strong.

About the desktop application version. In that case the chosen program language would be C/C++ [58] [59]. That is the most efficient solution and would have give a more robust software, although more complicated for the user to compile and execute the code. Still you can still provide the binaries for users how doesn't want to use the makefile.

11.3 Bad WSDL practices which required adjustments

As we saw in the section 9.3 in page 32 and figures 1, 5 and 6, programmers tend to encapsulate the inputs/outputs of an operation in only one part of the message. This is very bad because is make it more difficult to understand to somebody else who want to use your WS, and of course you always want your code to be readable for the next person who use it. In this case we had to compensate with shredding. This help the user to have a better visualization of the required input or output for an operation. Otherwise the user can understand quickly just by looking to the graphical representation of the operation what is going in or out of the operation. Variables names have meanings and are important. If an operation have input 'GetMatrixByNameRequest:Complex' we don't know what we are suppose to give. But with this correction that input transform into 'Name:String' , 'Format:String', 'Database:String' which is much more intuitive. Lets remind here that the W3 foundation discourage the use of Complex Types.

11.4 WSDL1.1 vs other options

A WSDL file is just one of many technologies out there to run a WS.

I chose WSDL 1.1 above WSDL 2.0 because that was what the department used most at that moment; and for example there were no EDAM annotations for WSDL 2.0.

In previous sections we talked about current technologies other that WSDL. We will now take a look on the disadvantages and why did I have to come up with something new.

Each technology description is a leap of faith. Everybody claim that the one they made are the best. However there are very few that gives real examples. Most of the time there is no algorithm behind the process, or no benchmarks for their implementation. The amount of tutorials is also very limited. So you need to spend a great deal of time studying a random technology, teaching yourself to use and hope for the best. In contrast, with the source code of this program you will find every documentation that I can think of which is going to be useful to somebody else in order to expand or understand how the program works. The problem when you study alternatives is the lack of proper documentation.

This is an overview of the most popular options. We talk about them in chapter 9.1 in page 30. In here we will review specific problems for each one:

OWL-S Semantic Markup for Web Services - The OWL Services Coalition With this approach you will need an external tool for linking the WDSL to your constructions. However using the EDAM approach the annotation are made in the WSDL itself without interfering with any previous declaration. So it is up to analyze the WSDL and see what it actually does rather than design something and hope for the best that you will actually find something that resemble your idea.

The Web Service Modeling Framework WSMF The problem with this technology is that is not implemented. Is just a theoretical approach and it haven't been tested at all. Also is highly focus on commercial WS and the complexity of the problems are limited; as in trying to solve which commerce you want is nearest to you, or how to make invoices between two companies be compatible. Alas, nothing as complex as for example trying to find an alternative protein which is cheaper to manufacture but does the same effect.

WSCL 1.0 In its own way, is similar to the XSD extensions. It doesn't really contribute to anything new and I view it as a failed attend to create a new standard to monopolized the technologies when this is not needed [62].

WS-BPEL The language only work on two services at the time so it only use to communicate between them and is not intended to analyze the semantic meaning behind them; or to analyze the significant of the entire workflow.

SWWS Aim to be the standard in the near future and it is probably the best option for the future. However as described before, there is nothing implemented yet and everything is pure theoretical.

So none of these give an actually solution for our goal. We want a workbench with nice functionality like those described in the introduction. And we want it to be able to analyze the semantics between services. Lets take a look to my proposed solution.

11.5 Open Source

There is a strong correlation in between the quality of the software and how free or open the software is [63] [64] [65]. There is a strong community behind software that become popular which improve them. Anybody who pick up my work should be able to upgrade without my help; thus eliminating the attachment of the original programmer to the future version or any other program that may spire.

My master is almost 100% financed by the Norwegian government. In my opinion, every public institution finance by public taxes should release all its findings with the most general public license possible and seek no other private funding.

During the making of this thesis I had to decide one browser for which the program will work for sure and be optimized. Based on the result of Acid Test, market share, operative system in which it runs and popularity of saying operative system, permissive license and open source philosophy, I decided to choose Chromium as the main browser for which I will develop the application.

Note that the software is optimized for Chromium, but it should work as well in Safari [14] since they both share the same rendering engine. I also made modification so it run in Firefox although I haven't test it extensively in that browser. The only part of the code you need to modify in order to make it work in a browser is the CSS file. CSS describe the rendering effects, so you need to specify the behavior for each engine. If you don't the application still works anyway, but you will find that the sidebar or the tool bar appears clunky and the elements have an estrange placement.

11.6 GUI

The Graphical User Interface of the program is a bit different from what a person might be used to. However the graphical engine is not attached to the kernel application and you can run the program even from the JavaScript console if you like. That also gives you the possibility of design your own graphical interface.

11.7 Future updates

You can automatically annotate WSDL with a brute force approach. Imagine you have a valid dataset of inputs and outputs for a set of ontologies that goes in an element. You can run the WS for every example of the dataset and compare the given output. If they match you have an element which you know the valid ontology. So in that case you will just add the SAWSDL field where correspond.

Is possible to expand the program in order to process different concepts from different ontologies simultaneously.

A minimap view for when you have a lot of images inside the workshop.

An ontology explorer that complement the tree view. Basically, if you click on an ontology box, its adjacent elements will pop out. The only reason of why

I didn't do this is because there is an overwhelming amount of data for the user already and you will need even a bigger screen.

A shortcut to show all active operation and bring them inside the current camera view.

Minimize the right column so you have a full screen canvas. You won't be able to add WSDL but you will have more space to design something with your current workshop state.

Allow users to manually annotate WSDL. Then the suggestions are send to the server and a human evaluate the most popular ones. If they are correct update the WSDL file with the given suggestion.

Allow user to rate WS. Some services are better than other, as in for example execution time, very few false negatives or positives, etc... So a workflow which could use operation A or operation B can decide which one to use based on users rating and experiences.

Save the workflow so another workbench can execute it. However there is still the problem with the translations between elements. And although it could be a good workflow idea according with the program it could be impossible to execute without some manual tweaks between inputs/outputs.

11.8 Summary

In this section we explained why did we make some decisions during the making of this project. We also talked about possible improvements to implement provided we had an infinite number of time. I feel like there are many possible expansions to this idea and I did my best to make an open source project with proper documentation so other people how likes it can expand it or modify it with their own vision, ideas, or visual design which they like most.

12 Conclusion

The program shows that in theory EDAM ontology is sufficient to annotate semantically WSDLs and allow for the automatic construction of workflows. With the exception of the Job ID problem, the program doesn't give you any false positives or false negatives. So if the program say that you can't link something is almost certain that you shouldn't try to do so. And with proper annotation, if the program tells you that you can connect to operations is because it makes sense to do it. Given this result I'm confident to conclude that WSDL-Workshop can give you every possible and correct combination of WSDL. This is a great breakthrough because you can let the computer do experiments automatically until you find one that gives you back the desire results, instead of having to search for it manually during days.

All aims were accomplished, except one that couldn't being tested. WSDL-Workshop tells if some data is compatible with another, using semantics and analyzing the datatypes. If an operation is compatible with another, using semantics. If one chain of operations makes more sense than other with the workflow score. It can easily be accessed by anyone with an Internet connection and a web browser; no installation or registration required. Anybody can modify the code or reproduce the results. It has a graphical user interface for the workflow construction, and an independent functionality that allows the user to discover and explore WSDLs by a variate of different filters.

I personally believe that the prototype was a success and that it can be easily made into a commercial version capable of competing with other similar programs like Galaxy or Taverna. Futhermore, it shows the potential use of semantics annotations to help professionals design experiments or to train students into understanding the concept of a workflow.

As stated in 8.4.7 in page 25 and the theory this program relies on a universal translator between two programs which are compatible but use different formats. Such system doesn't exist at the moment, although there are approximations focused on different topics. For example Benchling [66] or BioWord [67] focused on DNA sharing between WS and users.

There are also small issues with the EDAM. First is that at the moment very little annotated WSDL exists. Second, minor concepts that required a few modifications to work as described in the discussion of the EDAM ontology in section 11.1 in page 62. The Workflow Score is implemented as described; however very little combination of workflow are possible at the moment due the lack of WSDL semantically annotated. In the near future we can expect to draw a sufficiently populated scatter-plot that allow us to tell when a workflow is within expected parameters.

13 Appendices

13.1 GIT Package

In here I'm going to overview the important files which you can find in the source code at `https://github.com/rafanozal/WSDL-Workshop`.

/**Doc** Folder with documentation.

/**Res** Here you can find the images for the icons, the CBU logo and the ontologies files.

/**WSDLs** Here you can find all the default WSDL files. Note the file directory.xml; using this you don't need to give additional directory listing to your server.

/**XSD** These are all the schema references that you can find inside any WSDL file.

file.txt This is an empty file. The server save your workflow state in here and send it to the user when the user want to save his work. JavaScript doesn't allow writing or reading files directly for security reasons, so this adjustment is require.

help.html Contain a plain HTML version of this document. When the user clicks help, he is automatically redirected to the WSDL-Workshop section.

index.html Is the main page of the program.

readme.txt A text file that contain this very description.

style.css Is the CSS file for index.html

WSDLWorkshop.js Contain the JavaSctipt code.

13.2 Index

Index

14 References

References

[1] Jan Christian Bryne, Eivind Valen, Man-Hung Eric Tang, Troels Marstrand, Ole Winther, Isabelle da Piedade, Anders Krogh, Boris Lenhard, and Albin Sandelin. Jaspar, the open access database of transcription factor-binding profiles: new content and tools in the 2008 update. *Nucleic Acids Res*, 36(Database issue):D102–6, January 2008.

[2] Wikipedia. Dna binding site — wikipedia, the free encyclopedia. `http://en.wikipedia.org/w/index.php?title=DNA_binding_site&oldid=577518026`, 2013. [Online; accessed 14-November-2013].

[3] Clustalw2 - phylogeny. `http://www.ebi.ac.uk/Tools/phylogeny/clustalw2_phylogeny/`.

[4] K. Sivashanmugam, K. Verma, and A. Sheth. Discovery of web services in a federated registry environment. In *Web Services, 2004. Proceedings. IEEE International Conference on*, pages 270–278, 2004.

[5] Uddi/xml. `http://uddi.xml.org/`.

[6] Eyhab Al-Masri and Qusay H. Mahmoud. Investigating web services on the world wide web. In *WWW2008 Beijing, China. Web Engineering - Web Service Deployment*, pages 795–804, 2008.

[7] Tom Oinn, Matthew Addis, Justin Ferris, Darren Marvin, Martin Senger, Mark Greenwood, Tim Carver, Kevin Glover, Matthew R Pocock, Anil Wipat, and Peter Li. Taverna: a tool for the composition and enactment of bioinformatics workflows. *Bioinformatics*, 20(17):3045–54, November 2004.

[8] UK. School of Computer Science at the University of Manchester. Taverna. `http://www.taverna.org.uk`.

[9] Manchester Southampton, led by David De Roure Oxford in the UK, and Carole Goble. Myexperiment. `http://www.myexperiment.org/`.

[10] David De Roure, Carole Goble, and Robert Stevens. The design and realisation of the virtual research environment for social sharing of workflows. *Future Generation Computer Systems*, 25(5):561–567, 2009.

[11] Belinda Giardine, Cathy Riemer, Ross C Hardison, Richard Burhans, Laura Elnitski, Prachi Shah, Yi Zhang, Daniel Blankenberg, Istvan Albert, James Taylor, Webb Miller, W James Kent, and Anton Nekrutenko. Galaxy: a platform for interactive large-scale genome analysis. *Genome Res*, 15(10):1451–5, October 2005.

[12] Mathematics and Computer Science departments at Emory University. Galaxy. `https://usegalaxy.org/`.

[13] AAke Edlund, Maarten Koopmans, Zeeshan Ali Shah, Ilja Livenson, Frederik Orellana, Jukka Kommeri, Miika Tuisku, Pekka Lehtovuori, Klaus Marius Hansen, Helmut Neukirchen, and Ebba Hvannberg. Practical cloud evaluation from a nordic escience user perspective. In *Proceedings of the 5th International Workshop on Virtualization Technologies in Distributed Computing*, VTDC '11, pages 29–38, New York, NY, USA, 2011. ACM.

[14] H. Sagehaug, P. Venkataraman, A. Töpfer, K. Tekle, M. Kalaš, P. Sztromwasser, A. K. Stavrum, M. Dondrup, S. Subramanian, F. Roque, S. M. Hollup, I. Jonassen, K. Petersen, and P. Puntervoll. esysbio: an adaptable workbench for collaborative life science research. eSysbio, 2013.

[15] Uni Computing and the University of Bergen. esysbio. `http://esysbio.org/about`.

[16] Aleksi Kallio Taavi Hupponen Petri Klemelä Massimiliano Gentile Kimmo Mattila Ari-Matti Saren Mikael Karlsson Eija Korpelainen. Chipster. `http://chipster.csc.fi/`.

[17] M Aleksi Kallio, Jarno T Tuimala, Taavi Hupponen, Petri Klemelä, Massimiliano Gentile, Ilari Scheinin, Mikko Koski, Janne Käki, and Eija I Korpelainen. Chipster: user-friendly analysis software for microarray and other high-throughput data. *BMC Genomics*, 12:507, 2011.

[18] Michael Reich, Ted Liefeld, Joshua Gould, Jim Lerner, Pablo Tamayo, and Jill P Mesirov. Genepattern 2.0. *Nat Genet*, 38(5):500–1, May 2006.

[19] Jill P. Mesirov Principal Investigator Jon Bistline Jared Nedzel Peter Carr Michael Reich Barbara Hill Jim Robinson Dong-Keun Jang Cathy Stein Heidi Kuehn Thorin Tabor Ted Liefeld Pablo Tamayo Judith McLaughlin Helga Thorvaldsdóttir Marc-Danie Nazaire. Genepattern. `http://www.broadinstitute.org/cancer/software/genepattern/`.

[20] University of Bergen, Trondheim Tromso includes the Universities in Oslo, and Aas. Elixir. `http://www.bioinfo.no/elixir`.

[21] Jon C. Ison, Matús Kalas, Inge Jonassen, Dan M. Bolser, Mahmut Uludag, Hamish McWilliam, James Malone, Rodrigo Lopez, Steve Pettifer, and Peter M. Rice. Edam: an ontology of bioinformatics operations, types of data and identifiers, topics and formats. *Bioinformatics*, 29(10):1325–1332, 2013.

[22] Html. `http://www.w3schools.com/html/`.

[23] Xml. `http://www.w3schools.com/xml/`.

[24] Schema. `http://www.w3schools.com/schema/`.

[25] Soap. `http://www.w3.org/TR/soap/`.

[26] Wsdl. `http://www.w3.org/TR/wsdl`.

[27] Wsdl2.0. `http://www.w3.org/TR/wsdl20/`.

[28] Jaspar wsdl. http://129.177.120.189/cgi-bin/jaspar2010/jaspar_db.pl.

[29] Belgique. Laboratoire de Bioinformatique des Génomes et des Réseaux (BiGRe) Jacques van Helden Université Libre de Bruxelles. Position-specific scoring matrices (pssm). http://biologie.univ-mrs.fr/upload/p202/01.4.PSSM_theory.pdf.

[30] Ontology. http://www.w3.org/standards/semanticweb/ontology.

[31] Css. http://www.w3schools.com/css/.

[32] Javascript. http://www.w3schools.com/js/.

[33] Acid test. http://acid3.acidtests.org/.

[34] SRI International (editor) Mark Burstein BBN Technologies Jerry Hobbs USC Information Sciences Institute Ora Lassila Nokia Drew McDermott Yale University Sheila McIlraith University of Toronto Srini Narayanan International Institute of Computer Science Massimo Paolucci Carnegie Mellon University Bijan Parsia The MIND Laboratory of the University of Maryland at College Park Terry Payne University of Southampton Evren Sirin The MIND Laboratory of the University of Maryland at College Park Naveen Srinivasan Carnegie Mellon University Katia Sycara Carnegie Mellon University David Martin. Owl-s: Semantic markup for web services. http://www.w3.org/Submission/OWL-S/.

[35] Ruben Lara, Holger Lausen, Sinuhe Arroyo, Jos de Bruijn, and Dieter Fensel. Semantic web services: description requirements and current technologies. In *International Workshop on Electronic Commerce, Agents, and Semantic Web Services, In conjunction with the Fifth International Conference on Electronic Commerce (ICEC 2003)*,, 2003.

[36] D. Fensel and C. Bussle. The web service modeling framework wsmf. WSMF.

[37] Wscl1.0. http://www.w3.org/TR/wscl10/.

[38] *Web Services Business Process Execution Language Version 2.0.*

[39] University of Innsbruck Juan Miguel Gomez. Brief survey of swws and wsmf. In *SWWS*, 2003.

[40] Rusch Doris C. Mechanisms of the soul &#38#8211 tackling the human condition in videogames. In *Breaking New Ground: Innovation in Games, Play, Practice and Theory*. Brunel University, September 2009.

[41] Cc3.0. http://creativecommons.org/licenses/by-nc-sa/3.0/.

[42] Geany. http://www.geany.org/.

[43] Ubuntu. http://www.ubuntu.com/.

[44] Netbeans. https://netbeans.org/.

[45] Glassfish. https://glassfish.java.net/.

[46] Linux mint. http://www.linuxmint.com/.

[47] Gpl. http://www.gnu.org/copyleft/gpl.html.

[48] Lgpl. http://www.gnu.org/licenses/lgpl.html.

[49] Kile. http://www.kde.org/applications/office/kile/.

[50] Chromium. http://www.chromium.org/Home.

[51] Webkit. http://www.webkit.org/.

[52] Safari. http://www.apple.com/safari/.

[53] Emboss. http://www.ebi.ac.uk/Tools/emboss/.

[54] Jack Tang. Java native layer exploits going up. http://blog.trendmicro.com/trendlabs-security-intelligence/java-native-layer-exploits-going-up/.

[55] Anthony Joe Melgarejo. A new exploit kit in neutrino. http://blog.trendmicro.com/trendlabs-security-intelligence/a-new-exploit-kit-in-neutrino/.

[56] Christopher Budd. How the java security situation quietly got much worse. http://blog.trendmicro.com/java-security-situation-quietly-got-much-worse/?sf17098354=1.

[57] Gelo Abendan. Java 6 zero-day exploit pushes users to shift to latest java version. http://blog.trendmicro.com/trendlabs-security-intelligence/java-6-zero-day-exploit-pushes-users-to-shift-to-latest-java-version/.

[58] Benchmarks. http://attractivechaos.github.io/plb/.

[59] Perl, python, ruby, php, c, c++, lua, tcl, javascript and java comparison. http://raid6.com.au/~onlyjob/posts/arena/.

[60] Security Engineer Justin Schuh. Saying goodbye to our old friend npapi. http://blog.chromium.org/2013/09/saying-goodbye-to-our-old-friend-npapi.html.

[61] Jquery. http://www.w3schools.com/jquery/.

[62] Randall Munroe. Standards. http://xkcd.com/927/.

[63] Annual coverity scan report finds open source and proprietary software quality better than industry average for second consecutive year. Press release, Coverity, 2013.

[64] Donato Barbagallo and Chiara Francalanci. The relationship among development skills, design quality, and centrality in open source projects. In Susan Newell, Edgar A. Whitley, Nancy Pouloudi, Jonathan Wareham, and Lars Mathiassen, editors, *ECIS*, pages 2024–2035, 2009.

[65] Eugenio Capra, Chiara Francalanci, and Francesco Merlo. An empirical study on the relationship between software design quality, development effort and governance in open source projects. *IEEE Transactions on Software Engineering*, 34(6):765–782, 2008.

[66] Benchling. `https://benchling.com/`.

[67] Laura J Anzaldi, Daniel Muñoz-Fernández, and Ivan Erill. Bioword: a sequence manipulation suite for microsoft word. *BMC Bioinformatics*, 13:124, 2012.