**Project**
ID2208
Programming Web-Services

**Kim Hammar**
**Mallu**
Due Date: 6 March 2017

# Problem Statement

The given task was to automatically match different web services together in different ways. The matching was done by comparing input and outputs of operations in web services descriptions and matching them both syntactically and semantically. To match syntactically the edit-distance was used to calculate similarity. To match semantically the structural concepts of operations were compared by analysing the corresponding ontology's of the web services descriptions.

# Main problems and solutions

- *Parsing*: Parsing was done with the WSDL4j library which supplied the basic functionality for reading WSDL and SAWSDL files. In addition we developed our own parsing-API on top of the WSDL4J API just to fit our needs for this specific task. For example we added methods like: `getOperations()`, `getHighestLevelAnnotated()`, `getBasicElements()` etc. Marshalling the output was done with the JAXB library.
- *Syntactic matching*: Syntactic matching was done by comparing every web-service description in pairs and then in turn comparing every operation in pairs by looking at the input and output elements and matching on their corresponding types by calculating the edit-distance.
- *Semantic matching*: For semantic matching, annotated WSDL files and OWL files were used as input. The matching was similar to that of syntactic but with the difference that input/output elements were compared by taking the highest-level annotated elements and extracting the semantic classes and then finally comparing the semantic relations of the associated semantic classes.

## Syntactic Matching

The syntactic matches we found that exceeded the threshold were quite few. Since the edit-distance algorithm and the XML format for outputting matches were already given the main challenge for this task was to extract the necessary information from the WSDL files to be able to calculate the edit distance. For example the parsing library we used did not allow to extract types of elements very easy nor did it have methods to check if the type were a basic type or a complex type. Since we were supposed to only do matching on basic types we had to do some traversing of the WSDL files using the WSDL4j library to extract the necessary information. We considered simple types who used basic types as their base for the matching as well.

Example of a matching that our syntactic matcher finds and outputs from the input WSDL files:

```
<MacthedElement>
    <OutputElement>Reference</OutputElement>
    <InputElement>Reference_No</InputElement>
    <Score>0.8464817248906141</Score>
</MacthedElement>
```

### Semantic Matching

Compared to the syntactic matching, the semantic matching found a lot more matches from the SAWSDL files that were given as input. Just like for the syntactic matching the main exercise here was to extract the semantic classes for the elements to be able to apply the methods from the given `MyOntManager.java`.

For instance this is how we checked if elements/types were annotated or not:

```
public boolean isAnnotated(Element element) {
        return element.hasAttribute("sawsdl:modelReference");
    }
```

Example of matching that our semantic matcher finds and outputs from the input SAWSDL files (0.6 means that the semantic class of the output element is a subclass of the semantic class of the input element):

```
<MacthedElement>
    <OutputElement>AuthorType</OutputElement>
    <InputElement>AgentType</InputElement>
    <Score>0.6</Score>
</MacthedElement>
```

## Conclusions

The practical part of this project assignment was mainly about understanding how to use the given code-base and to develop some parsing functionality to be able to read the input WSDL and SAWSDL files and apply the edit-distance algorithm for syntactic matching and using the `MyOntManager.java` to find the semantic relationships between classes based on the information in the OWL files.

What we can conclude from the different types of matching-techniques is that adding very basic structural information through annotations can give you a lot richer ways to search/match/reason about data without any previous knowledge, which isn't possible by using traditional syntactic approaches. For instance with simple key-word searching or using the edit-distance approach as in this project, two services, one that is a gift-shop for birthdays and one that is a Swedish poison-dealer shop might both have operations called "getGift()" (gift is Swedish for poison) and appear as perfect syntactic matches while in reality the services are completely different, i.e their semantics are different. Of course humans can easily distinguish two semantically different things but machines can't. Adding semantics to the web in a machine-readable format by using technologies like OWL is a necessary step to achieve the vision of the future web as being autonomous and not needing human supervision to the same degree.

## Attachments

Documented source code can be found in the attached zip-file. See README.MD in the root directory for instructions how to execute and build the program.