Querying WSDL Repositories with Grok

SCHOOL OF Computing
Queen's UNIVERSITY
Douglas H. Martin, James R. Cordy, Thomas R. Dean

Introduction

Web Service Description Language (WSDL) is the language used to specify web service interfaces, but they are often generated and read automatically and thus are not always easy to read or understand. In this project, **we attempt to use TXL and Grok to extract facts from WSDL files and query those facts to answer questions about WSDL descriptions.**

Motivation

In our previous work, we used TXL (a source transformation system) to restructure WSDL operations into self-contained units that inserted the referred elements into the elements that refer to them. This made it possible to perform analysis on them using clone detection and Latent Dirichlet Allocation (LDA) to find meaningful relationships between service operations. The problem is that it is **very tedious to look up the results** to see what they have in common, and **difficult to find similar operations before analysis in order to calculate the recall** of these approaches.

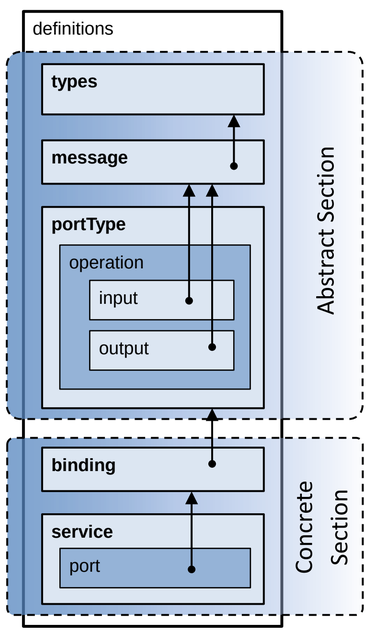
Goals

The goal of this project is to **provide a means of answering the common questions that arise when working with WSDL**. Some example questions include:

- "What types do these operations have in common?"
- "What is the difference between these operations (i.e. what parameters are different)?"
- "Which operations use this type?"

Web Service Description Language (WSDL)

A WSDL description starts with a service, which provides one or more operations. Each of these operations contains inputs, outputs and faults. These have "message" attributes associated with them that refer to message elements. Message elements contain parts, which can behave in one of two ways. First, it may have an "element" attribute, which refers to an element in an XML Schema at the beginning of the description. Second, it may have a "type" attribute that refers to a type that may be defined in the XML Schema. Either way, the elements in the XML Schema can contain other elements that have types defined elsewhere, and so on. **The disjointedness of WSDL descriptions make them difficult to understand, which makes it even more difficult to ask questions about them when you have many in a repository.**



Grok

Grok is a system for manipulating factbases of binary relations (e.g. "relation X Y") with its own language to do so. It is **similar to a relational database**, but optimized to handle large numbers of facts (hundreds of thousands) and perform transitive closure operations.

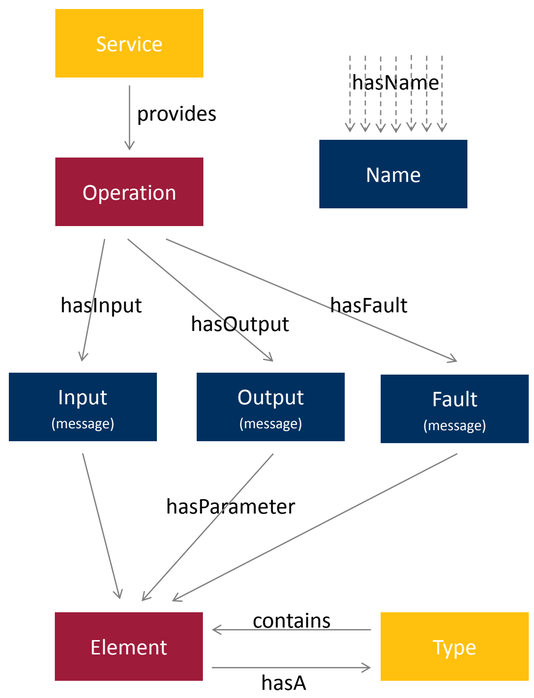
Some **common Grok operators** are:

- Union (+) — Combines two sets (e.g. {A, B} + {C, D} = {A, B, C, D})
- Intersection (^) — Returns the elements two sets have in common (e.g. {A, B, C} ^ {B, D, E} = {B})
- Composition (o) — Joins two sets of relations (e.g. R1 = {(A, B), (C, D)}, R2 = {(B, E)}, R1 o R2 = {(A, E)})
- Projection (.) — Separates a set from a relation (e.g. Right: {(A, B), (B, C), (A, C)} . {"C"} = {B, A}) (e.g. Left: {A} . {(A, B), (B, C), (A, C)} = {B, C})
- Transitive Closure (+) — Computes all inferred facts in a relation (e.g. R = {(A, B), (B, C), (C, D)}, R+ = {(A, B), (A, C) (A, D), (B, C), (B, D), (C, D)})

Method

First, we **created a middle model of WSDL** that focuses on the parts in which we're most interested and removes some unnecessary layers (e.g. inputs/outputs and message elements are combined into one unit). Then, to construct the facts, **we used TXL to extract the names of each entity** in the service (the boxes in the diagram) **and create the relations between them** (the arrows). We were able to re-use most of the work we had done previously.

Now that the facts are constructed, all we have to do is **load them into Grok** and it is **ready for querying**.



Model of WSDL used to construct facts for Grok (the arrows form the binary relations)

Conclusion

Using Grok and the facts that we extract from a repository of WSDL files, **we are able to answer all of the question we set out to ask** (and more):

- *"What types do these operations have in common?"*

```
params := hasInput + hasOutput
types := (params o hasParameter o hasA o hasName)
Op1_types := {"<Op1>"} . types
Op2_types := {"<Op2>"} . types
Op1_types ^ Op2_types
```

- *"What is the difference between these operations (i.e. what parameters are different)?"*

```
(Op1_types - Op2_types) + (Op2_types - Op1_types)
```

- *"Which operations use this type?"*

```
types . {"<type>"}
```