Consolidating WSDL Descriptions of Web Services

Douglas Martin James R. Cordy

School of Computing, Queen's University
Kingston, Canada

Abstract

A challenge for the Smart Internet will be the automated tagging of equivalent or similar services, both in terms of domain semantics and service protocols, in support of efficient discovery and selection of relevant alternative services for the current matters of concern. Code similarity detection is an established technique that can be brought to bear on this problem if service descriptions can be partitioned into appropriate units for comparison. Unfortunately, specifications written in Web Service Description Language (WSDL) are poorly structured for this purpose, with relevant information for each service operation scattered widely over WSDL service descriptions. In this work we describe a first step in leveraging code similarity techniques to identify and tag similarities in WSDL services. Using source transformation techniques, we describe a method for reorganizing WSDL descriptions such that they are both more human readable and better suited to analysis by similarity detection tools.

1 Introduction

An important part of the Smart Internet initiative is the automated tagging and classification of web services that are similar in domain or service protocol [1]. Code similarity tools, or clone detectors [4] provide a mature, scalable similarity detection technology that can be leveraged to assist in this problem. Our aim is to leverage these existing techniques to analyze WSDL service descriptions for similarities that can then be more easily classified and tagged. However, WSDL service description files usually contain unified descriptions of all of the operations that the web service has to offer, with pieces of each operation scattered over different parts of the file. This not only makes the operation descriptions difficult to read, but it also presents a challenge for clone detection tools. In this paper we address this problem by utilizing source transformation in TXL [3] to restructure WSDL service descriptions to consolidate and separate complete standalone operation descriptions, which can then be analyzed by clone detection techniques to cluster them into sets of semantically and structurally similar operations.

2 Background

Some research has been done to find related services in a repository of WSDL descriptions. Syeda-Mahmood et al. [2] have explored the use of domain-independent and domain-specific ontologies when comparing service descriptions. They found that on a large service repository this allowed them to obtain more relevant results. The method outlined in this paper may provide even better results when used in conjunction with theirs, because it allows the comparison of individual operations as opposed to whole service descriptions, supporting analysis of views of services rather than only entire service descriptions.

3 Methodology

WSDL service description files contain several sections: *types*, which define a schema with elements and type declarations (or complexTypes) used later in the file; *messages*, which contain parts associated with elements described in the types section; *portTypes*, which contain operations with inputs/outputs associated with messages; *bindings*, which contain a protocol for the messages associated with each portType; and *services*, which contain a set of ports for communicating with the web service. Each of these contains a part of the information necessary to understand each service operation – to understand any individual operation, it is necessary to chase down all the relevant pieces from the other sections. In this first experiment we ignore bindings and services, since they add no new information and are unique to the particular web service.

Our tool uses the source transformation system TXL [3] to construct a set of consolidated service operations from a WSDL description. It works by merging the pieces together to form a set of independent consolidated operation descriptions that clearly and directly identify inputs, outputs and possible faults.

Phase 1: Extract Operations
The first step in the consolidation is to set up the skeleton of each operation. This acts as a base in which to put the elements found later in the types section. We extract the operations from the portTypes section, which contain the operation's input, output, and faults. Figure 1 shows an example of such an operation from a hotel reservation system.

Phase 2: Inject Messages
Each of the operation's inputs, outputs, and faults refers to a message that appears elsewhere in the description. The next step is to find that message and inject it into each corresponding tag, giving something like what is shown in Figure 2.

Phase 3: Inject Element into Parts
Each part of a message has a particular element associated with it, described in the types section. We find and inject this element into the part tag, resulting in something like Figure 3.

Phase 4: Consolidate Elements
For each element in the newly constructed part, we find the type declaration of its type, if one exists. We then take the elements of that type and insert them into the parent element. The same process is repeated recursively for all the elements of the type, until all elements in the part are native XML types (eg. string, int, etc.). In the end, the result looks something like Figure 4.

Phase 5: Clean Up
Some unnecessary tags (e.g. `<complexType>`, `<sequence>`, and so on) remain after phases 1-4, so next we remove these to make the operation definitions cleaner and easier to read. The final result for our hotel reservation example is shown in Figure 5, and the entire process can be visualized as illustrated in Figure 6. What we're left with is a clear human-readable standalone representation of each operation, its required input and expected output. For example, we can see that the *ReserveRoom* operation in Figure 5 takes a *Payment* object and a *Room* object and returns an acknowledgment. Further, from the composition of the *Payment* and *Room* objects, we can easily tell what information is required for this service.

```
<operation name="ReserveRoom">
  <input message="tns1:ReserveRoomRequest"/>
  <output message="tns1:ReserveRoomResponse"/>
  <fault message="tns1:RoomNotAvailableException"/>
</operation>
```

Figure 1 *Phase 1*

```
<operation name="ReserveRoom" >
  <input message="tns1:ReserveRoomRequest">
    <message name="ReserveRoomRequest">
      <part name="body"
            element="xsd1:ReserveRoomRequest"/>
    </message>
  </input>
  <output message="tns1:ReserveRoomResponse">
    <message name="ReserveRoomResponse">
      <part name="body"
            element="xsd1:ReserveRoomResponse"/>
    </message>
  </output>
  <fault message="tns1:RoomNotAvailableException">
    <message name="RoomNotAvailableException">
      <part name="body"
            element="xsd1:RoomNotAvailableException"/>
    </message>
  </fault>
</operation>
```

Figure 2 *Phase 2*

```
<operation name="ReserveRoom" >
  <input message="tns1:ReserveRoomRequest">
    <message name="ReserveRoomRequest">
      <part name="body"
            element="xsd1:ReserveRoomRequest">
        <element name="ReserveRoomRequest">
          <complexType>
            <sequence>
              <element name="payment"
                       type="tns1:Payment"/>
              <element name="room"
                       type="tns1:Room"/>
            </sequence>
          </complexType>
        </element>
      </part>
    </message>
  </input>
  . . .
</operation>
```

Figure 3 *Phase 3*

4 Preliminary Results

Early results from similarity detection tools show promise. When comparing operations from WSDL descriptions with no consolidation, the results were sparse and unusable. However, after our restructuring many interesting groups of similar operations were identified (e.g., operations related to employee information, shipping information, customer information, and so on).

5 Conclusion

WSDL descriptions of web services pose problems for finding similarities. We have developed a way to make descriptions both more human readable and amenable to code similarity techniques.

```
<operation name="ReserveRoom" >
  <input message="tns1:ReserveRoomRequest">
    <message name="ReserveRoomRequest">
      <part name="body"
            element="xsd1:ReserveRoomRequest">
        <element name="ReserveRoomRequest">
          <complexType>
            <sequence>
              <element name="payment"
                       type="tns1:Payment">>
                <element name="ccNumber"
                         type="xsd:int"/>
                <element name="cardHolder"
                         type="xsd:string"/>
                <element name="expiryDate"
                         type="xsd:date"/>
                <element name="PIN"
                         nillable="true"
                         type="xsd:int"/>
              </element>
              <element name="room"
                       type="tns1:Room">
                <element name="roomID"
                         type="xsd:int"/>
                <element name="numBeds"
                         type="xsd:int"/>
                <element name="isSmoking"
                         type="xsd:boolen"/>
              </element>
            </sequence>
          </complexType>
        </element>
      </part>
    </message>
  </input>
  . . .
</operation>
```

Figure 4 *Phase 4*

Acknowledgements

The authors thank Scott Grant for the inspiration behind this idea. This work was supported in part by an IBM Faculty Award, and by NSERC.

About the Authors

Douglas Martin is a first year M.Sc. student and James R. Cordy is a professor in the School of Computing at Queen's University. They can be reached by email at {doug,cordy}@cs.queensu.ca .

References

[1] J.W. Ng, M. Chignell, J.R. Cordy, "The Smart Internet: Transforming the Web to Fit User Needs", Proc. *CASCON 2009*, Nov. 2009 (to appear).

[2] T. Syeda-Mahmood, G. Shah, R. Akkiraju, A. Ivan, R. Goodwin, "Searching Service Repositories by Combining Semantic and Ontological Matching", Proc. *ICWS 2005*, July 2005, pp. 13-20.

[3] J.R. Cordy, "The TXL Source Transformation Language", *Sci. Comput. Program.* 61(3), 2006, pp. 190-210.

[4] C.K. Roy, J.R. Cordy and R. Koschke, "Comparison and Evaluation of Code Clone Detection Techniques and Tools: A Qualitative Approach", *Sci. Comput. Program.* 74(7), 2009, pp. 470-495.

```
<operation name="ReserveRoom" >
  <input message="tns1:ReserveRoomRequest">
    <message name="ReserveRoomRequest">
       <part name="body"
             element="xsd1:ReserveRoomRequest">
         <element name="ReserveRoomRequest">
           <element name="payment"
                    type="tns1:Payment">
             <element name="ccNumber"
                      type="xsd:int"/>
             <element name="cardHolder"
                      type="xsd:string"/>
             <element name="expiryDate"
                      type="xsd:date"/>
             <element name="PIN"
                      nillable="true"
                      type="xsd:int"/>
           </element>
           <element name="room"
                    type="tns1:Room">
             <element name="roomID"
                      type="xsd:int"/>
             <element name="numBeds"
                      type="xsd:int"/>
             <element name="isSmoking"
                      type="xsd:boolean"/>
           </element>
         </element>
       </part>
    </message>
  </input>
  <output message="tns1:ReserveRoomResponse">
    <message name="ReserveRoomResponse">
       <part name="body"
             element="xsd1:ReserveRoomResponse">
         <element name="ReserveRoomResponse"/>
       </part>
    </message>
  </output>
  <fault message="tns1:RoomNotAvailableException">
    <message name="RoomNotAvailableException">
      <part name="body"
            element="xsd1:RoomNotAvailableException">
        <element name="RoomNotAvailableException"/>
      </part>
    </message>
  </fault>
</operation>
```

Figure 5 *Phase 5*

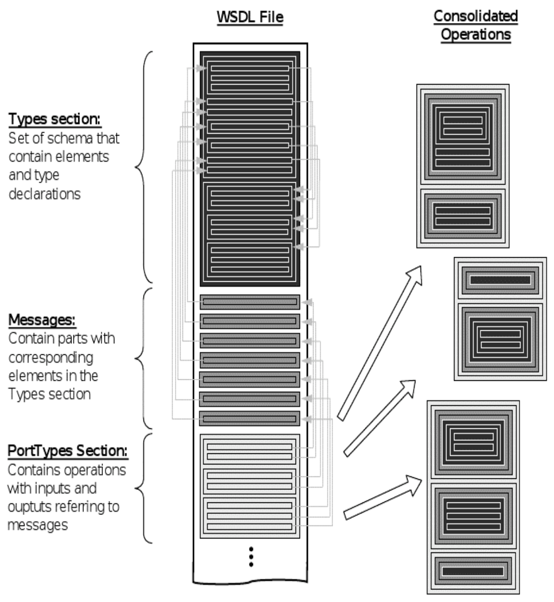
WSDL File
Consolidated Operations
Types section: Set of schema that contain elements and type declarations
Messages: Contain parts with corresponding elements in the Types section
PortTypes Section: Contains operations with inputs and ouptuts referring to messages

Figure 6 *Visualization of the Restructuring*