Analyzing Web Service Similarity Using Contextual Clones

Douglas Martin
School of Computing, Queen's University
Kingston, Ontario, Canada
doug@cs.queensu.ca

James R. Cordy
School of Computing, Queen's University
Kingston, Ontario, Canada
cordy@cs.queensu.ca

ABSTRACT

There are several tools and techniques developed over the past decade for detecting duplicated code in software. However, there exists a class of languages for which clone detection is ill-suited. We discovered one of these languages when we attempted to use clone detection to find similar web service operations in service descriptions written in the Web Service Description Language (WSDL). WSDL is structured in such a way that identifying units for comparison becomes a challenge. WSDL service descriptions contain specifications of one or more operations that are divided into pieces and intermingled throughout the description. In this paper, we describe a method of reorganizing them in order to leverage clone detection technology to identify similar services. We introduce the idea of *contextual clones* – clones that can only be found by augmenting code fragments with related information referenced by the fragment to give it context. We demonstrate this idea for WSDL and propose other languages and situations for which contextual clones may be of interest.

Categories and Subject Descriptors

D.2.7 [**Distribution, Maintenance, and Enhancement**]: Restructuring, reverse engineering, and reengineering; D.2.5 [**Testing and Debugging**]: Testing tools

General Terms

Measurement, Experimentation

Keywords

Clone detection techniques, Web services, WSDL

1. INTRODUCTION

The web is rapidly moving toward a service-oriented architecture. Many web applications today provide APIs to call their services that can be used to create new applications and compose new and more complex services. One way they do this is using Web Service Description Language (WSDL) [2] to specify how to invoke the operations that the service provides, and make them available for other applications to use. Tagging similar descriptions is an important part of both service discovery and identification of alternative services when a service experiences downtime. Due to the rapid growth of the Web, manual tagging is impractical and the infrastructure of the future will require automation of this task. While there has been previous work in detecting similarities in WSDL [7, 16, 17], in our work we instead propose to leverage clone detection techniques, which provide a mature and scalable means to discover similarities. However, the scattered syntax and semantics of WSDL service descriptions makes it difficult to identify appropriate units of comparison, and simple clone detection on WSDL operation descriptions does not provide any useful answers.

The WSDL description of a web service contains specifications of one or more operations that the service provides. At the heart of the service description is the `<portTypes>` element, which contains a list of operations and the inputs, outputs and faults handled by each. However, the input and output elements rarely provide any valuable information by themselves (in most cases, their names are the same as the operation name with "Request" or "Response" appended to the end). Their purpose is to reference other elements of the WSDL description that provide type information and describe the parameters the operation expects and what can be expected in return.

This scattered referential form makes it difficult for a clone detector to identify appropriate units (potential clones) for comparison. Comparing entire WSDL descriptions doesn't allow us to identify similarity at the operation level; if two services share a similar operation but the remainder of the services is different enough, a clone detector may ignore it, yielding a low recall level. On the other hand, if we compare at the operation description level, we ignore the type and parameter information described elsewhere in the service description, yielding a high level of false positives.

To illustrate this, consider the two *GetStock* operations taken from two different service descriptions, shown in Figure 1. Looking at these operations, we might conclude that they are clones; in fact, we would probably agree that these are exact clones. However, by

```
<operation name="GetStock" >
   <input message="tns:GetStockRequest" />
   <output message="tns:GetStockResponse" />
</operation>

<operation name="GetStock" >
   <input message="tns:GetStockRequest" />
   <output message="tns:GetStockResponse" />
</operation>
```

Figure 1. Two GetStock operations.

looking at the rest of the service descriptions in which they are embedded – or *contextualizing* the operations – we see that the use of the word "stock" has two completely different meanings in the two services. The first service uses "stock" to mean inventory, while the second uses "stock" to mean a financial stock on the stock market. Thus ignoring contextual information can lead us to falsely identify operations as clones.

In this paper, we introduce the idea of *contextual clones* – those that can only be identified by augmenting code fragments (potential clones) with referenced contextual information, and show how we have used them to obtain more meaningful and useful results from clone detection on WSDL service descriptions. Contextual clones are clones found by consolidating de-localized code into unified fragments more appropriate as potential clones for a clone detector. This helps avoid false positive situations like the one above, but can also uncover new clones that we may never have found otherwise. In the remainder of this paper, we discuss how to contextualize WSDL code fragments in Section 2, and Section 3 explains how we have used them to find similar operations in WSDL service descriptions. Section 4 presents some possible future applications of contextual clones, and Section 5 gives a brief overview of related work and how it differs from our approach.

2. CONTEXTUALIZING CODE FRAGMENTS

In this section, we describe how we can contextualize code fragments for comparison. Specifically, we describe how we contextualize WSDL operations, and why it is necessary.

2.1 Contextual Clones

Before we go further, let us formally define what we mean by a contextual clone. Contextualized code is created by expanding parts of a code fragment to include information referenced elsewhere; giving meaning to something that may be relatively meaningless otherwise. These newly expanded fragments are used as potential clones and given to a clone detector. The clones found in this way are called contextual clones.

Definition 1: *Contextualized Code*. **Contextualized code** is a code fragment that has been modified or expanded to include information referenced elsewhere.

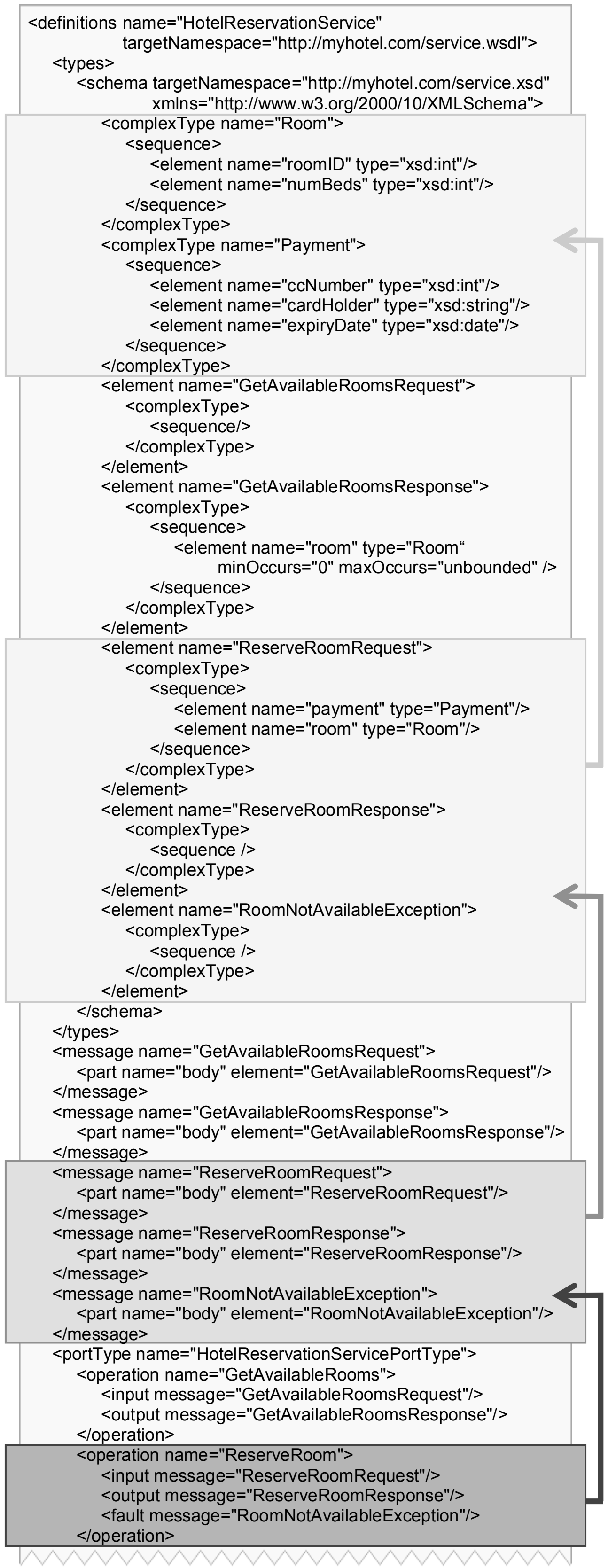
Definition 2: *Contextual Clone*. A **contextual clone** is a code clone found by comparing contextualized code fragments as potential clones.

Contextual clones are rarely the result of copying and pasting, but rather indicate higher level relationships between fragments.

2.2 Contextualizing WSDL Operations

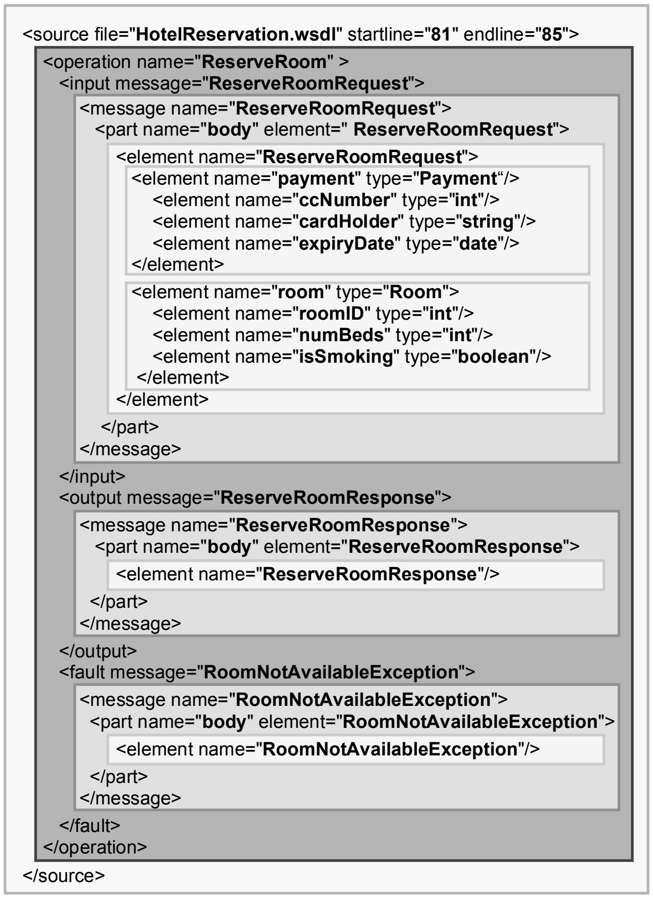
In our initial attempts to use clone detection to find similarities in WSDL descriptions of web services [10], we quickly discovered that, as an XML-based language, it was not organized in such a way that we could easily extract operation descriptions for comparison as potential clones.

The description of an operation in a WSDL document is divided into many pieces. It begins in the `<portTypes>` section where



```
<definitions name="HotelReservationService"
            targetNamespace="http://myhotel.com/service.wsdl">
  <types>
    <schema targetNamespace="http://myhotel.com/service.xsd"
              xmlns="http://www.w3.org/2000/10/XMLSchema">
      <complexType name="Room">
        <sequence>
          <element name="roomID" type="xsd:int"/>
          <element name="numBeds" type="xsd:int"/>
        </sequence>
      </complexType>
      <complexType name="Payment">
        <sequence>
          <element name="ccNumber" type="xsd:int"/>
          <element name="cardHolder" type="xsd:string"/>
          <element name="expiryDate" type="xsd:date"/>
        </sequence>
      </complexType>
      <element name="GetAvailableRoomsRequest">
        <complexType>
          <sequence/>
        </complexType>
      </element>
      <element name="GetAvailableRoomsResponse">
        <complexType>
          <sequence>
            <element name="room" type="Room“
                minOccurs="0" maxOccurs="unbounded" />
          </sequence>
        </complexType>
      </element>
      <element name="ReserveRoomRequest">
        <complexType>
          <sequence>
            <element name="payment" type="Payment"/>
            <element name="room" type="Room"/>
          </sequence>
        </complexType>
      </element>
      <element name="ReserveRoomResponse">
        <complexType>
          <sequence />
        </complexType>
      </element>
      <element name="RoomNotAvailableException">
        <complexType>
          <sequence />
        </complexType>
      </element>
    </schema>
  </types>
  <message name="GetAvailableRoomsRequest">
    <part name="body" element="GetAvailableRoomsRequest"/>
  </message>
  <message name="GetAvailableRoomsResponse">
    <part name="body" element="GetAvailableRoomsResponse"/>
  </message>
  <message name="ReserveRoomRequest">
    <part name="body" element="ReserveRoomRequest"/>
  </message>
  <message name="ReserveRoomResponse">
    <part name="body" element="ReserveRoomResponse"/>
  </message>
  <message name="RoomNotAvailableException">
    <part name="body" element="RoomNotAvailableException"/>
  </message>
  <portType name="HotelReservationServicePortType">
    <operation name="GetAvailableRooms">
      <input message="GetAvailableRoomsRequest"/>
      <output message="GetAvailableRoomsResponse"/>
    </operation>
    <operation name="ReserveRoom">
      <input message="ReserveRoomRequest"/>
      <output message="ReserveRoomResponse"/>
      <fault message="RoomNotAvailableException"/>
    </operation>
```

Figure 2. The pieces of the ReserveRoom operation from a hotel reservation service description are highlighted.



```
<source file="HotelReservation.wsdl" startline="81" endline="85">
  <operation name="ReserveRoom" >
    <input message="ReserveRoomRequest">
      <message name="ReserveRoomRequest">
        <part name="body" element=" ReserveRoomRequest">
          <element name="ReserveRoomRequest">
            <element name="payment" type="Payment"/>
              <element name="ccNumber" type="int"/>
              <element name="cardHolder" type="string"/>
              <element name="expiryDate" type="date"/>
            </element>
            <element name="room" type="Room">
              <element name="roomID" type="int"/>
              <element name="numBeds" type="int"/>
              <element name="isSmoking" type="boolean"/>
             </element>
          </element>
        </part>
      </message>
    </input>
    <output message="ReserveRoomResponse">
      <message name="ReserveRoomResponse">
        <part name="body" element="ReserveRoomResponse">
          <element name="ReserveRoomResponse"/>
        </part>
      </message>
    </output>
    <fault message="RoomNotAvailableException">
      <message name="RoomNotAvailableException">
       <part name="body" element="RoomNotAvailableException">
          <element name="RoomNotAvailableException"/>
       </part>
      </message>
    </fault>
  </operation>
</source>
```

Figure 3. The contextualized code fragment for the ReserveRoom operation of the simple hotel reservation service description in Figure 1.

operations are listed in their own `<operation>` element. Each of these elements contain a number of `<input>`, `<output>` or `<fault>` tags that correspond to a `<message>` element defined somewhere else in the description. These in turn contain `<part>` elements that may refer to other remote elements in the `<types>` section. The elements in the `<types>` section can also contain elements that have other types associated with them, which can in turn contain more elements, and so on. The result is that a number of different operation descriptions may be split into remote pieces that are intermingled in the same description. Figure 2 shows an example WSDL service description of a simple hotel reservation service with the *ReserveRoom* operation highlighted and its parts traced through the file.

In attempting to use clone detection to uncover similarity among services, the problem we faced was what to use as code fragments. We could use the entire service description, but that would not provide the granularity to compare services at the operation level. It would also mean that two services with a single similar operation might be ignored if the remaining operations were different enough. To achieve a finer level of granularity, we could use the `<operation>` elements in the `<portTypes>` section. The problem with this is that, while it gives us the name of the operation, it ignores all of the associated parameter information from the `<types>` section. This difficulty led us to develop the idea of contextual clones.

To implement contextual clone detection, we used TXL [3], a source transformation language, to generate contextualized potential clones for comparison. Beginning with a WSDL grammar, we created a set of transformation rules that turn WSDL service descriptions into a set of localized, reorganized operation descriptions that can then be used as potential clones. Beginning with the `<operation>` elements and working its way down the hierarchy recursively, the transformation copies elements inside the elements that reference them. The result is a set of self-contained operation descriptions that not only provide context to the original operations, but also give them structure and make them easier to read and understand. The contextualized fragment is then wrapped in XML `<source>` tags that give the name of the file from which it was taken and the beginning and end lines of the original operation description so that it can be traced back. Figure 3 shows an example of a contextualized WSDL operation for the *ReserveRoom* example operation from the hotel reservation service description in Figure 2.

3. DETECTING CONTEXTUAL CLONES

Once we have a complete set of contextualized code fragments, we can use them as potential clones to identify contextual clones using any off-the-shelf clone detector. For our experiment, we chose the NiCad clone detector [13], which uses an efficient and scalable hybrid parsing and text comparison technique based on the longest common subsequence (LCS) algorithm to identify "near-miss" clones, those fragments that are very similar but perhaps not identical. NiCad uses a plugin architecture that allowed us to add support for WSDL and contextual clones as a language plugin that NiCad treated as simply another language parser / extractor.

3.1 Contextual Clones in WSDL

It is important to clarify at this point that the operation "clones" we find in WSDL, even the exact ones, may not indicate actual identical web service operations. Instead, they indicate that the operations process and produce the same type of data, which tells us that they are related in some way. Even if they appear to be exactly the same, there is no way to tell for sure whether they *do* the same thing. This is because WSDL is a description language, not a programming language, and does not give us access to the server side implementation code for operations, but rather describes how they can be invoked and combined.

Unfortunately it is difficult to get access to server side code for web services, mainly because most of it is proprietary. But in any case, including it in our comparisons would defeat the point of using WSDL. Also, our goal is the ability to identify similar services and operations to be able to tag and match them appropriately, not necessarily to find identical operations.

3.2 Detecting Contextual Clones in WSDL

We experimented with contextual clones in WSDL using two sets of web service descriptions (some of which are proprietary and cannot be reproduced here). *Set 1* consists of more than 200 web services containing over 1,100 operations, many of which are very similar or duplicates. *Set 2* consists of more than 500 services containing over 7,500 operations from a wide variety of domains, obtained using the web services search engine at Seekda [18], the world's largest repository of public web services.

Using NiCad, we analyzed the clones in each set. The limit on the minimum clone size had to be decreased from the default 5 to 3 lines since the smallest non-contextualized operations in WSDL can be as small as 3 lines (opening and closing operation tags and at least one input/output/fault tag inside).

For each set, we compared the results from NiCad at various near-miss difference thresholds on the set of contextualized operations against the set of original non-contextualized operations (i.e. the operation elements in the `<portTypes>` element). The results are summarized in Figure 4. The first table shows the number of code clones found (the number of fragments for which there exists at least one clone), while the second shows the number of clone classes (the number of groups of clones). Looking at the tables, we see that Set 1 had a much larger proportion of clones than Set 2, which was expected because there were a large number of duplicated WSDL descriptions in it.

We can observe a number of other interesting things in the tables as well. First, the number of clones almost always decreases when we consider contextualized clones, which shows that contextualized clones allow us to filter out false positives found by using only non-contextualized operations. When we look more closely at these cases, we find that many non-contextualized operations that appear to be clones actually turn out to be very different when their parameters are expanded.

For example, let's consider the eailer example of the *GetStock* operations. Figure 5 shows the two operations before and after contextualization. At first glance (before contextualization), they

a)

Difference Threshold	Set 1		Set 2	
	No Context	Context	No Context	Context
0.0	852	705	1434	1066
0.1	852	734	1434	1228
0.2	879	775	1438	1637
0.3	884	813	1469	1637

b)

Difference Threshold	Set 1		Set 2	
	No Context	Context	No Context	Context
0.0	169	187	587	433
0.1	169	139	587	499
0.2	172	142	589	631
0.3	171	136	591	631

Figure 4. The results of NiCad near-miss clone analysis of the contextualized operations of two different sets of web services. (a) shows the number of code (operation) clones found, and (b) shows the number of clone classes.

would appear to be the same operation, and if we use standard clone detection on these fragments, they would be considered exact clones. However, if we expand the `<input>` and `<output>` tags to include the other elements to which they refer, we see that they are actually from two very different domains; one refers to "stock" as in "inventory", wheras the other refers to "stock" as in a financial stock. Depending on the difference threshold used, these may not be considered clones at all, let alone exact clones.

The second, and most important, thing we notice is that we can also find clones that we wouldn't have found otherwise. Take, for example, the set of operations in Figure 6. All of these operations

Non-Contextualized:
```
<operation name="GetStock" >
  <input message="tns:GetStockRequest" />
  <output message="tns:GetStockResponse" />
</operation>
```

Contextualized:
```
<operation name="GetStock" >
  <input message="tns:GetStockRequest">
    <message name="GetStockRequest">
      <part name="parameters" element="tns:GetStockRequest">
        <element name="GetStockRequest">
          <element name="InventoryNumber" type="xsd:int" />
        </element>
      </part>
    </message>
  </input>
  <output message="tns:GetStockResponse">
    <message name="GetStockResponse">
      <part name="parameters" element="tns:GetStockResponse">
        <element name="Stock">
          <element name="Supplier" type="xsd:string"/>
          <element name="Warehouse" type="xsd:string"/>
          <element name="OnHand" type="xsd:string"/>
          <element name="OnOrder" type="xsd:string"/>
          <element name="Demand" type="xsd:string"/>
        </element>
      </part>
    </message>
  </output>
</operation>
```

Non-Contextualized:
```
<operation name="GetStock" >
  <input message="tns:GetStockRequest" />
  <output message="tns:GetStockResponse" />
</operation>
```

Contextualized:
```
<operation name="GetStock" >
  <input message="tns:GetStockRequest">
    <message name="GetStockRequest">
      <part name="parameters" element="tns:GetStockRequest">
        <element name="GetStockRequest">
          <element name="symbol" type="xsd:string" />
        </element>
      </part>
    </message>
  </input>
  <output message="tns:GetStockResponse">
    <message name="GetStockResponse">
      <part                                   name="parameters"
element="tns:GetStockResponse">
        <element name="Stock">
          <element name="date" type="xsd:string"/>
          <element name="open" type="xsd:float"/>
          <element name="high" type="xsd:float"/>
          <element name="low" type="xsd:float"/>
          <element name="close" type="xsd:float"/>
          <element name="volume" type="xsd:float"/>
        </element>
      </part>
    </message>
  </output>
</operation>
```

Figure 5. Two GetStock operations appear to be exact clones when non-contextualized, but are actually from different domains.

```
<wsdl:operation name="GetRealChartCustom">
   <wsdl:input message="tns:GetRealChartCustomSoapIn"/>
   <wsdl:output message="tns:GetRealChartCustomSoapOut"/>
</wsdl:operation>

<wsdl:operation name="GetLastSaleChartCustom">
   <wsdl:input message="tns:GetLastSaleChartCustomSoapIn"/>
   <wsdl:output message="tns:GetLastSaleChartCustomSoapOut"/>
</wsdl:operation>

<wsdl:operation name="DrawHistoricalChartCustom">
   <wsdl:input message="tns:DrawHistoricalChartCustomSoapIn"/>
   <wsdl:output message="tns:DrawHistoricalChartCustomSoapOut"/>
</wsdl:operation>

<wsdl:operation name="DrawIntraDayChartCustom">
   <wsdl:input message="tns:DrawIntraDayChartCustomSoapIn"/>
   <wsdl:output message="tns:DrawIntraDayChartCustomSoapOut"/>
</wsdl:operation>

<wsdl:operation name="GetDelayedChartCustom">
   <wsdl:input message="tns:GetDelayedChartCustomSoapIn"/>
   <wsdl:output message="tns:GetDelayedChartCustomSoapOut"/>
</wsdl:operation>

<wsdl:operation name="GetTopicChartCustom">
   <wsdl:input message="tns:GetTopicChartCustomSoapIn" />
   <wsdl:output message="tns:GetTopicChartCustomSoapOut" />
</wsdl:operation>

<wsdl:operation name="GetTopicBinaryChartCustom">
   <wsdl:input message="tns:GetTopicBinaryChartCustomSoapIn"/>
   <wsdl:output message="tns:GetTopicBinaryChartCustomSoapOut"/>
</wsdl:operation>

<wsdl:operation name="DrawRateChartCustom">
   <wsdl:input message="tns:DrawRateChartCustomSoapIn"/>
   <wsdl:output message="tns:DrawRateChartCustomSoapOut"/>
</wsdl:operation>

<wsdl:operation name="DrawRateChartCustom">
   <wsdl:input message="tns:DrawRateChartCustomSoapIn"/>
   <wsdl:output message="tns:DrawRateChartCustomSoapOut"/>
</wsdl:operation>

<wsdl:operation name="DrawYieldCurveCustom">
   <wsdl:input message="tns:DrawYieldCurveCustomSoapIn"/>
   <wsdl:output message="tns:DrawYieldCurveCustomSoapOut" />
</wsdl:operation>
```

Figure 6. A cluster of contextual clones related to charting operations.

relate to charting, but the only indication that they may be related is the word "chart" in their names (except for the last one). A clone detector might not identify these as clones at all, because their operation and message names are different and, except for the common WSDL syntax, they share nothing. However, when contextualized (not shown here), we quickly see that they share very similar data types and elements, with only a few changes between them.

4. FUTURE WORK

Thus far, we have described our approach to finding similar web service operations, but have ignored the original goal of tagging them. In our continuing research, we are exploring the use of domain ontologies to infer appropriate service tags for the contextual clones we find in WSDL service descriptions.

In the course of this research, we have also come to believe that the notion of contextual clones could have uses in other languages, specifically in other domain-specific XML-based languages. We are particularly interested in exploring the use of them to find clones in models represented in XML-based textual exchange representations. For example, XMI, the exchange language for UML class diagrams, seems to be a likely candidate.

5. RELATED WORK

There are many tools and techniques to identify code clones [14], but while many different normalizations have been used in clone detection, to our knowledge no one has modified source code in this way before searching for clones. Other work on clone detection for the Web has either been focused on finding clones in static web pages [5, 6, 11], server-side scripts [12] or client-side scripts [1, 9] rather than web services.

Clone detection for web services may be new, but a number of people have used other techniques to find similarities among service operations.

Dong et al. [7] developed a web service search engine called Woogle that gives the user the ability to perform a similarity search. Once a user finds a web service that is close to meeting their needs, they may search for services that are similar to it, take similar inputs, or compose well with the given service. At the heart of this search is a clustering algorithm that groups the service's parameters into semantically similar concepts. This clustering algorithm uses the heuristic that parameters that occur together often tend to express the same concepts.

Syeda-Mahmood et al. [17] have explored the use of domain-independent and domain-specific ontologies when comparing service descriptions. Specifically, they looked at large company mergers or acquisitions where each company has its own set of web services that do similar things, but use different terminology. They used a number of techniques to aid in the search. First, they used word tokenization to separate multi-term parameter names (e.g. PartNumber) into its individual terms (e.g. PartNumber becomes "Part" and "Number"). Then, they used part-of-speech tagging and filtering to identify noun phrases and adjectives. Next, they expanded abbreviations (e.g. Cust becomes Customer). Finally, they used a synonym search with a thesaurus like WordNet to find synonyms for each word and assigned a similarity score based on how close the words were. They then use a matching algorithm to produce a ranked list of matching services and tested it using a domain-independent ontology, a domain-specific ontology, and none at all. What they found was that using a domain-specific ontology improved precision over a domain-independent ontology.

Stroulia et al. [16] developed a suite of methods designed to aid developers in the search for a suitable web service operation. They implemented 3 methods for this. First, when only a textual description of a service is available, they use a vector-space model to match the description with the text inside the service's `<documentation>` tags. A variation of this method uses WordNet to find synonyms (words with similar meaning), hypernyms (word parent), hyponyms (word children), and sibling senses (e.g. am, are, is) for the textual descriptions and apply scores based on how close they match. Second, when a stub of a web service is available and a structurally similar service is desired, they do structure matching. In this method, they compare data types, messages and operations of all pair-wise combinations from the source and target services. Finally, when a stub of a web service is available and a semantically similar service is desired, they do semantic structure matching. This method is an extension of the structure matching described above except instead of looking for compatible type mappings to find a syntactically similar service, they look for semantically compatible mappings to try to find a semantically similar service. This suite of methods solves the problem of service discovery based on different stages in a development process.

There has been research into finding clones in models as well. Girschick [8] used a difference algorithm to detect changes in UML class diagrams, and Störrle [15] used a query-based approach to detect similarities in UML models. There has also been work done using a graph-based approach with MatLab/Simulink models [4]. None of these approaches, however, uses a standard text-based code clone detection technique like NiCad.

6. CONCLUSION

As the web continues to grow and web applications get more sophisticated, an efficient and automatic method for discovering new services becomes more important. Clone detection is a mature field that can be leveraged to assist this problem, and we believe we have successfully shown that it can be with some modifications to the WSDL descriptions.

The great thing about our approach is that it can be used by any number of tools to detect similarities. New tools can even be created that are specifically designed to take advantage of contextualization or web services.

7. ACKNOWLEDGMENTS

This work was made possible by an IBM Center for Advanced Studies (CAS) Fellowship and Faculty Award, and is supported in part by the Natural Sciences and Engineering Research Council of Canada (NSERC).

8. REFERENCES

[1] Calefato, F., Lanubile, F. and Mallardo, T. 2004. Function Clone Detection in Web Applications: A Semiautomated Approach. *Journal of Web Engineering* 3,1, 3-21.

[2] Christensen, E., Curbera, F., Meredith, G. and Weerawarana, S. 2001. Web Services Description Language (WSDL) 1.1. *World Wide Web Consortium (W3C)*. Available at: http://www.w3.org/TR/wsdl.

[3] Cordy, J.R. 2006. The TXL Source Transformation Language. *Science of Computer Programming*. 61,3, 190-210.

[4] Deissenboeck, F., Hummel, B., Juergens, E., Schätz, B., Wagner, S., Girard, J.F., Teuchert, S. 2008. Clone Detection in Automotive Model-Based Development. In *Proceedings of the 30th International conference on Software Engineering (ICSE '08)*, Leipzig, Germany, May 2008, 603-612.

[5] Di Lucca, G. A., Di Penta, M., Fasilio, A. R., and Granato, P. 2001. Clone analysis in the web era: An approach to identify cloned web pages. In *Proceedings of 7th IEEE Workshop on Empirical Studies of Software Maintenance*, Florence, Italy, November 2001, 107-113.

[6] Di Lucca, G.A., Di Penta, M., Fasolino, A.R. 2002. An approach to identify duplicated web pages. In *Proceedings of 26th COMPSAC International Computer Software and Applications Conference*, Oxford, England, August 2002, 481-486.

[7] Dong, X., Halevy, A., Madhaven, J., Nemes, E. and Zhang, J. 2004. Similarity Search for Web Services. In *Proceedings of the 30th VLDB Conference*, Toronto, Canada, August 2004, 372-383.

[8] Girschick, M. 2006. Difference Detection and Visualization in UML Class Diagrams. *Technical Report TUD-CS-2006-5*. TU Darmstadt.

[9] Lanubile, F. and Mallardo, T. 2003. Finding Function Clones inWeb Applications. In *Proceedings of the 7th European Conference on Software Maintenance and Reengineering (CSMR'03)*, Benevento, Italy, March 2003, 379-386.

[10] Martin, D. and Cordy, J. R. 2010. Towards Web Services Tagging by Similarity Detection. In *The Smart Internet Current Research and Future Applications*, M. Chignell, J. Cordy, J. Ng, Y. Yesha Eds. Springer, New York, NY, USA, 216-233.

[11] Rajapakse, D. C., and Jarzabek, S. 2005. An Investigation of Cloning in Web Applications. In *Proceedings of the 5th Intl Conference on Web Engineering (ICWE'05)*, Sydney, Australia , July 2005, 252-262.

[12] Rajapakse, D.C. and Jarzabek, S. 2007. Using Server Pages to Unify Clones in Web Applications: A Trade-Off Analysis. In *Proceedings of the 29th international conference on Software Engineering (ICSE '07)*, Minneapolis, MN, USA, May 2007, 116-126.

[13] Roy, C.K. and Cordy, J.R. 2008. NICAD: Accurate Detection of Near-Miss Intentional Clones Using Flexible Pretty-Printing and Code Normalization. In *Proceedings of the 16th IEEE International Conference on Program Comprehension (ICPC 2008)*, Amsterdam, The Netherlands, June 2008, 172-181.

[14] Roy, C.K., Cordy, J.R. and Koschke, R. 2009. Comparison and Evaluation of Code Clone Detection Techniques and Tools: A Qualitative Approach. *Science of Computer Programming*. 74,7, 470-495.

[15] Störrle, H. 2010. Towards clone detection in UML domain models. In *Proceedings of the 4th European Conference on Software Architecture: Companion Volume (ECSA '10)*, C. Cuesta Ed. ACM, New York, NY, USA, pp. 285-293.

[16] Stroulia, E. and Wang, Y. 2005. Structural and Semantic Matching for Assessing Web Service Similarity. *International Journal of Cooperative Information Systems* 14,4, 407-437.

[17] Syeda-Mahmood, T., Shah, G., Akkiraju, R., Ivan, A. and Goodwin, R. 2005. Searching Service Repositories by Combining Semantic and Ontological Matching. In *Proceedings of the 3rd International Conference on Web Services (ICWS 2005)*, Orlando, FL, USA, July 2005, 13-20.

[18] Seekda, 2009. Web Services Search Engine. http://webservices.seekda.com/.