WSCells for the Personal Web

Douglas Martin James R. Cordy

School of Computing, Queen's University, Kingston, Ontario

Abstract

In our previous work [1], we developed WSRD – a tool that analyzes WSDL service descriptions to gather related pieces of operations into self-contained units and put them into a standalone description we call a Web Service Cell, or WSCell (pronounced "wizzle"). WSCells are complete, self-contained single service operations that can be analyzed, compared and (in theory) used independently of their original services. In this paper we explore the role that WSCells may be able to play in the vision of the Personal Web.

1 Introduction

Web services described using the Web Service Description Language (WSDL) contain descriptions of multiple operations that are co-mingled throughout the service description. The interpretation of an individual operation depends on both its context in the description and the many remote pieces, such as types, portTypes and bindings, on which it depends. Isolating a particular operation description for analysis or service composition can be a tedious task.

In our previous work [1], we developed a tool we call WSRD (Web Service Restructuring of Descriptions) that collects and consolidates the information in a WSDL file related to each operation and packages them into concise self-contained descriptions that we call Web Service Cells, or WSCells (pronounced "wizzles") for short. WSCells provide a finer granularity view of services by separating operation descriptions from the description of the entire web service.

WSCells were originally designed to aid in the analysis of web service similarity and support web service tagging automation. However, in the context of the Personal Web, we now believe that they may have a more interesting role in aiding the selection and composition of personalized web services. WSRD can be used to separate WSDL descriptions into their constituent operation WSCells, and then users or advanced algorithms can pick and choose the operations that work best for any given situation, environment or current matter of concern, and compose them with each other to form a custom, personalized new service (Figure 1).

In this paper, we provide a brief background of our work on WSRD (Section 2), as well as some new ways that we envision WSCells being used to compose personal services (Section 3).

2 Background

In our previous work, our original plan was to modify WSDL descriptions to be better suited for processing using clone detection methods, with

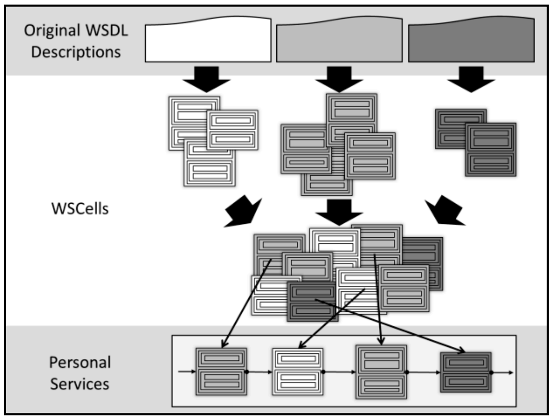

Figure 1
Separation of WSDL services into WSCells, and re-combination of WSCells into Personal Services

the goal of being able to identify and tag similar web service operations [1]. In this section, we briefly describe this technique using an illustrative example.

2.1 The Problem

Each WSDL file contains a description of one or more operations provided by a web service. Each of these descriptions is context-dependent and broken into several parts throughout the file, making it difficult to pick out and understand the entire description of a particular operation.

The complete description of an operation begins in the *portTypes* section of the WSDL file, where its inputs, outputs and possible faults are declared. Each of these refers to a *message* element, which in turn contains *parts* referring to elements in the *types* section. The types section contains an XML Schema that defines structured data types used for passing operation parameters. An element may have a type defined in the *types* section that contains elements of other types also defined there, and so on.

Thus the complete set of relevant pieces of the WSDL service description describing a particular operation are scattered throughout the WSDL file, and mixed in with irrelevant pieces of other operation descriptions. This makes it difficult to perform any sort of analysis of individual operations, or even of the service itself. This is where WSRD comes in.

2.2 WSRD

Using TXL [3], a source transformation language, we constructed a set of rules that transform the WSDL description of a web service into a set of self contained operation descriptions for the operations provided by the service. We call these self-contained operation descriptions Web Service Cells, or WSCells (pronounced “wizzles”) for short, because they are the like the cells that together form a web service. An example of one such WSCell is shown in Figure 2. Here we see an operation called *ReserveRoom* that takes a *Payment* object and a *Room* object as input and outputs an acknowledgement, with a fault in the case the room is unavailable.

WSRD includes a set of scripts that automates the generation of the complete set of WSCells for all WSDL files in a directory. Extracted WSCells can either be represented in a single file, or separated into individual files.

```
<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 2
Sample Web Service Cell (WSCell) generated by WSRD

3 WSCells for the Personal Web

While the original intention of WSRD was to make clone detection of WSDL descriptions possible, WSCells can be useful in other applications as well. The self contained nature of WSCells make them great for analyzing, comparing and mix-and-matching operations, and for storing all information necessary to understand and use an operation separately from the entire service.

There are 3 main ways in which we envision WSCells being used in composing personal services: *imperative composition* is driven entirely by the user; *declarative composition* is driven by the user’s goals; and *self-driven composition* is driven entirely by the system as a cellular automaton. Each of these three possibilities is illustrated in Figure 3.

3.1 Imperative Composition

Because WSCells contain all the information needed to use a web service operation, they can be thought of as beads or blocks that can be piled or strung together like a child’s blocks. A user can

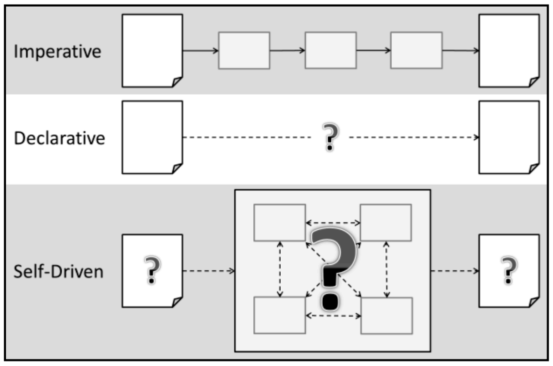

Figure 3
Three kinds of personal service composition

easily, given the right tools, pick and choose WSCells and hand compose them in a way that suits their particular needs to form a personalized service. Repetition, forking and joining structures can be imagined to form more complex combinations of operations. This is similar to the idea of "scripting" service composition in a higher-level notation.

As a concrete example, consider a generalization of Yahoo Pipes [2] where, instead of boxes that transform data streams like RSS or Atom, the boxes represent web service operations (WSCells) from a variety of web services. Users can connect the outputs of one operation to the inputs of another or, for a constant value, connect a text box with a value to an input. An example of this can be seen in Figure 4. In the figure, WSCells are represented using a combination of block notation from Yahoo Pipes and classes from a UML class diagram.

3.2 Declarative Composition

WSCells could also be automatically combined according to a user's goals, using a declarative approach to personal service composition that would iterate through the set of available WSCells looking for a path from the currently known information to the user's desired goal information. The implementation would be much like that of Prolog [4] – using unification and backtracking technology to "solve" the problem, without the necessity of the user explicitly specifying how to do so.

WSCells work well in this situation, because they give a concise, self-contained representation of the structure of a web service operation that can easily be matched or connected

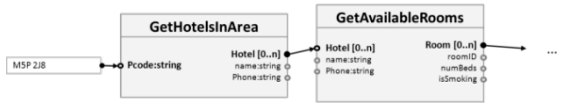

Figure 4
Example interface for imperative composition

to others. More specifically, they give a clear representation of the operation's inputs and outputs complete with their types and constraints. This makes it easy to match and explore whether or not two operations can be composed, in a way similar to the way Prolog unifies terms. All information about input and output for a particular operation is contained directly in its WSCell, meaning there is no need to search the entire WSDL description, and potential combinations can be discovered and explored efficiently.

3.3 Self-Driven Composition

While imperative and declarative composition rely on user input for composing services, self-driven composition relies on a cellular automaton model to discover new and interesting compositions to offer to the user. In this model, WSCells become active participants, looking for "partners" that they can combine with in meaningful ways to form new, higher-level "molecular" WSCells which can in turn look for others to combine with, and so on, gradually discovering the set of combined services that they can offer the user.

When combined with constraints to identify "interesting" combinations and reject uninteresting ones, this model of "emergent" service combination discovery could be interesting for the future.

4 Conclusion and Future Work

We have shown how our tool WSRD, or more specifically the self-contained WSCells that it produces, could prove to be a useful new concept in the discovery and composition of personalized services for the Personal Web. Their self-contained nature make WSCells ideal for selecting and combining operations from range of services, either imperatively by direct user selection, or automatically using either declarative or emergent programming to achieve user goals.

We have stated that WSCells contain all of the information that is needed to use a web service operation; however this is only partially true, and more work remains to be done if our idealistic vision is to be achieved. For example, it can be seen from Figure 2 that WSCells do not contain the address of the web service to which they belong. The reason is that this information was deemed unnecessary for the original purpose of finding similarities and was purposely left out. Future work will be put into packaging elements of the description that were ignored in our original work to make it possible to call the web service using only the operation's WSCell.

Acknowledgements

This research was made possible by an IBM CAS Fellowship and is partially supported by NSERC.

About the Authors

Douglas Martin is a second 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] D. Martin and J.R. Cordy, Chapter 12. "Towards Web Services Tagging by Similarity Detection" in M. Chignell et al. (eds.), *The Smart Internet: Current Research and Future Applications*, Lecture Notes in Computer Science 6400, Springer Verlag, October 2010, pp. 222-240.

[2] Yahoo Inc., "Pipes: Rewire the Web." Yahoo Pipes. *http://pipes.yahoo.com/pipes/.*

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

[4] P. Deransart, L. Cervoni, and A. Ed-Dbali, *Prolog: the Standard: Reference Manual*, Springer Verlag 1996.