EXPERIMENTAL STUDY ON THE USE OF SEMANTIC WEB CONCEPTS FOR MATCHING AND ASSEMBLING RICH CLIENT COMPOSITE APPLICATIONS

THESIS

Presented to the Graduate Council of
Texas State University-San Marcos
in Partial Fulfillment
of the Requirements

for the Degree

Master of SCIENCE

by

Michael Pierre Carlson, B.S.

San Marcos, Texas
May 2008

EXPERIMENTAL STUDY ON THE USE OF SEMANTIC WEB CONCEPTS

FOR MATCHING AND ASSEMBLING RICH CLIENT

COMPOSITE APPLICATIONS

Committee Members Approved:

Anne Ngu, Chair

Xiao Chen

Rodion Podorozhny

Approved:

J. Michael Willoughby
Dean of the Graduate College

COPYRIGHT

By

Michael Pierre Carlson

May 2008

DEDICATION

To my wonderful parents, Pierre and Marlene Carlson, for whom I have no words to express my thanks for everything they have helped me to achieve.

To my amazing wife and life partner Holly Riedelbach. I have loved you from the beginning, over ten years now. I could not imagine my life without you.

ACKNOWLEDGEMENTS

I would like to thank the chair of my thesis committee, Dr. Ngu of the Computer Science Department of Texas State University-San Marcos. Her continued assistance, guidance, and review allowed me to complete this work. Additionally, I would like to thank the other members of my committee, Dr. Chen and Dr. Podorozhny for their review and assistance with this work.

I also would like to express my appreciation to Kristen Riedelbach, friend and sister-in-law, for her help in reviewing and editing this thesis. Kristen has given much of her personal time reviewing my many drafts.

Finally, I wish to thank my amazing wife Holly. She has been wonderfully understanding of the time I have spent over the last few years working on this degree. We have certainly missed out on a lot of opportunities over the years because of my work. Without her support, I would not have been able to finish.

TABLE OF CONTENTS

LIST OF FIGURES

FIGURE **PAGE**

ABSTRACT

EXPERIMENTAL STUDY ON THE USE OF SEMANTIC WEB CONCEPTS

FOR MATCHING AND ASSEMBLING RICH CLIENT

COMPOSITE APPLICATIONS

by

Michael Pierre Carlson, B.S.

Texas State University-San Marcos

May 2008

SUPERVISING PROFESSOR: ANNE NGU

Composite applications are a line of business applications constructed by connecting, or wiring, disparate software components into combinations that provide a new level of function to the end user without the requirement to write any new code. The components that are used to build a composite application are generally built within a Service Oriented Architecture (SOA). Many of the first SOA platforms exclusively relied on web services (WSDL-based) as components in the composite application. With

the emergence of new standards, such as OSGi, the components used in composite applications have grown to include more than just web services. Components can be built from web applications, portlets, native widgets, legacy software, and Java technologies.

One of the most widely distributed SOA platforms is the Eclipse Rich Client Platform (RCP), which is built on the OSGi standard. Eclipse and OSGi provide a means for developers to create components in an SOA environment and declaratively extend the function of existing components. IBM Lotus Expeditor extends Eclipse's RCP by providing a composite application framework, which allows end users to construct composite applications from separately developed components and declaratively wire those components so that they can communicate and execute together. This provides for new levels of functionality not available in any of the individual components and provides the end user with the ability to customize their applications to meet their business needs.

There are challenges to combining components into composite applications, especially when components are developed at different times, by different groups, using different technologies, naming conventions, and structures. Similar to web services' UDDI registry, a catalog of components could contain many separately developed components that use different naming conventions. A given enterprise may have hundreds of similar components available for reuse in a catalog, but manually searching and finding compatible and complementary components could be a tedious and time-consuming task. Additionally, none of the existing SOA environments, including RCP and server based implementations, provide a way to leverage the search techniques that have been developed to assist the user in locating compatible non-web service components for composite applications. Unlike web services, many components used in RCP have graphical user interfaces built from technologies such as portlets, Eclipse Views, and native application windows. Depending on the technology used or the type of user interface being presented, certain components may not be valid for use in a particular composite application. For example, in a portal based environment, such as BEA WebLogic Portal, only portlet based user interfaces would be valid selections when assembling a composite application. Discerning this could be a difficult process up front, or could result in repeated cycles of trial and error, especially when the target environment supports a variety of technologies.

The main contributions of this thesis are as follows. First it is shown that existing techniques, technologies, and algorithms used for finding and matching web service components (WSDL-based) can be reused, with only minor changes, for the purpose of finding compatible and complementary non-web service based components for composite applications. These components may include graphical user interfaces, which are not an artifact in web service components. By building on the techniques initially developed for web services matching, the problems associated with finding useful and valid components for composite applications using high level concepts is possible. This enables the progressive construction of composite applications from a catalog of available components without deep knowledge of the components in the catalog. Second, it will be shown that the additional characteristics of non-web service based components, specifically graphical user interface details, can be categorized, described, and matched in a similar fashion to the programmatic inputs of the components. Though similar in some

respects to web services, these additional characteristics of a component allow for further match processing logic to be used to provide better results for the novice user when searching for components to integrate into the composite application. Finally, it will be shown using sample applications and scenarios that by taking into account the unique characteristics of an RCP composite application (i.e. coexistence of user interface components), new techniques of merging descriptions can be leveraged to provide better results when searching for new components to add to the composite application.

Chapter I

INTRODUCTION

1.1 Composite Applications

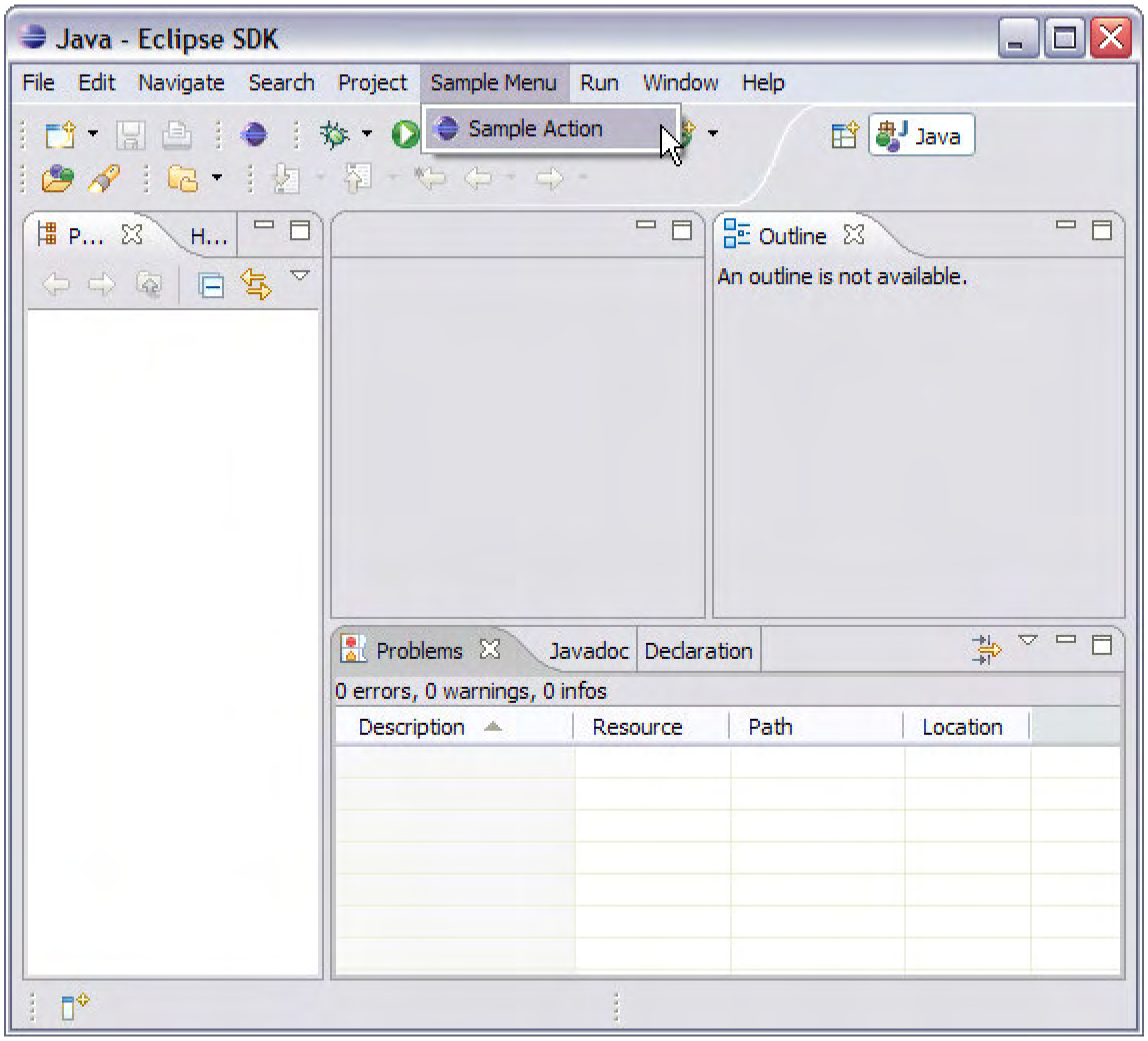
Composite applications are a line of business applications constructed by connecting, or wiring, disparate software components into combinations that provide a new level of function to the end user without the requirement to write any new code. Generally deployed on a Service Oriented Architecture (SOA) platform, the various components are often built without knowledge of the other components with which they will interact at runtime. Components in most server-based SOA platforms are generally headless – meaning they provide no graphical user interface. Popular component models include web services, Enterprise Java Beans (EJB), and Common Object Request Broker Architecture (CORBA).

Each of these technologies provides methods of describing their Application Programming Interface (API) to developers. Using these APIs, developers can build applications that make use of the services provided by the components. EJBs are specific to the Java programming language and therefore the programmatic interface is generally specified using a Java Interface (e.g. Home and Remote Interface). For web services, the implementation of the web service, be it Java, .NET, C++, etc. is abstracted from the API. The API is described using Web Services Description Language [1] (WSDL), which is a format of Extensible Markup Language [2] (XML). Graphical User interface components (menus, buttons, navigation links which are generally known as GUI) are typically decoupled from EJB and web services specifications. Work has been done to create tools that can create a basic client user interface from a web service WSDL file [3]. There have also been several standards developed for web services by the World Wide Web Consortium (W3C) and other standards organizations. For these reasons, interoperability between web services is greater than interoperability between any preceding systems. This standardization has led to many additional standards and technologies for describing, finding, and matching web services.

The components of RCP composite applications as discussed in this work are more than just headless, server side logic components. In this context, components generally contain a user interface, built from technologies such as JSPs and HTML, Eclipse Standard Widget Toolkit (SWT), Swing, native windowing systems, etc. Like web services and EJBs, components can take programmatic inputs and provide programmatic outputs. In our components, programmatic inputs will generally cause changes in the graphical user interface, and user interaction with the graphical user interface will cause the programmatic outputs to be fired. An example of this type of component is a Portlet [4], though components are not limited to this technology or the Java programming language.

1.2 Eclipse and SOA

The Eclipse [5] Rich Client Platform (RCP) provides a SOA environment for building client side applications. Starting with Eclipse 3.0, an implementation of the OSGi [6] specification has been used as the core framework for building separately manageable components in Eclipse. In OSGi, the smallest manageable piece of code is a bundle. For historical reasons, Eclipse uses the term “plug-in” instead of the term “bundle,” though for all intents and purposes these terms are equivalent. In addition to the SOA capabilities that are defined by OSGi, Eclipse also provides an extension point framework. This framework allows components to extend the capabilities of other components. For example, the Eclipse workbench (an example workbench is shown in Figure 1) is the primary controlling user interface component. In order to add a menu item to the workbench the developer will define an extension to the org.eclipse.ui.actionSets extension point. The developer can define the characteristics of the item, such as the category to add the item to, the label for the item, and the action to take when the item is selected by the user. In this example, the Sample Menu item and Sample Action item have been added to the standard workbench.


Java - Eclipse SDK
File Edit Navigate Search Project Sample Menu Run Window Help
Sample Action
Java
P... H...
Outline
Problems Javadoc Declaration
Description Resource Path Location

An outline is not available.

0 errors, 0 warnings, 0 infos

Figure 1: Eclipse IDE Workbench with Sample menu item

The extension point mechanism provides a clean way of declaratively extending the functionality of a component (the workbench in this example), though it does require knowledge of the component and extension point at development time. This required build time knowledge is a limitation that can affect the ways in which a component can be used, and further restrict their usage in RCP composite applications. For example, if the component being extended, say the Outline view shown in Figure 1, is not available in the final deployment environment, then the new component that extends the Outline view may not run, even if it is providing other services not related to the Outline view. So while the extension point method can remove some of the internal dependencies between components, it does not completely solve the problem of building loosely coupled components for deployment in an SOA environment.

1.3 Lotus Expeditor

The IBM Lotus Expeditor [7] platform extends the Eclipse Rich Client Platform with an additional framework called the Composite Application Infrastructure (CAI). A high level architecture diagram for Lotus Expeditor including the Lotus Notes plug-in is shown in Figure 2.

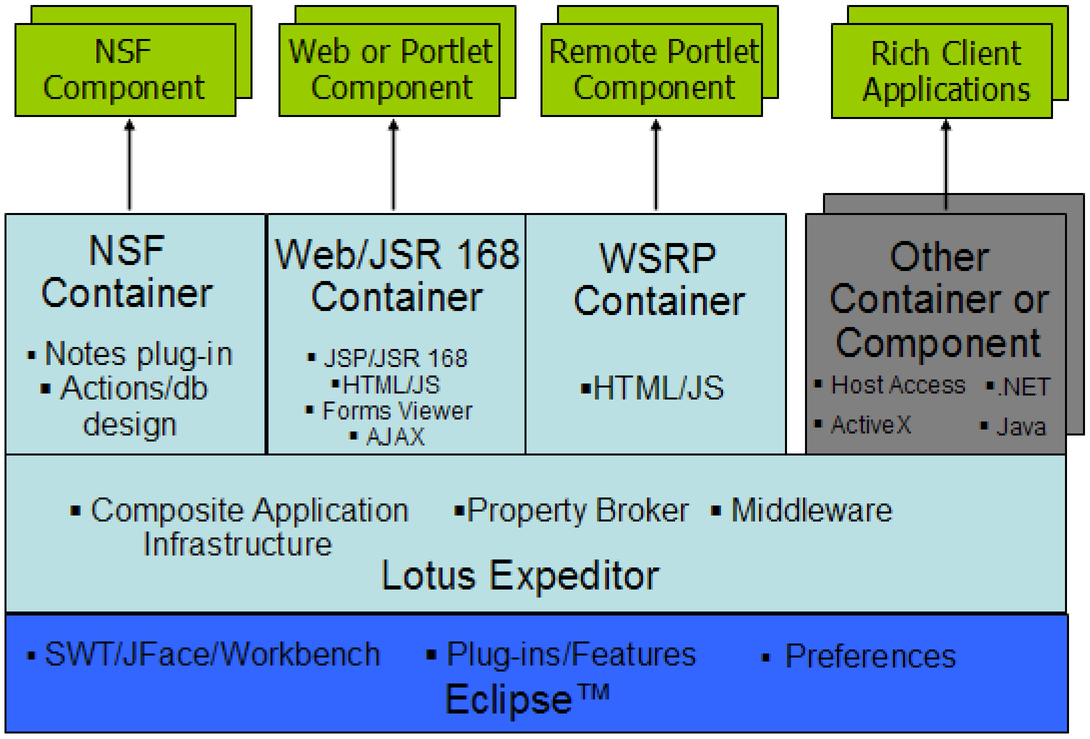


Figure 2: Architecture of Lotus Expeditor with Lotus Notes plug-in

Figure 3 shows the Lotus Expeditor Client default workbench and several sample applications installed under the Open button. Using a custom workbench and set of extension point, the look has been significantly changed from the Java IDE workbench shown in Figure 1, though the same extension point concepts are used to build this user interface.

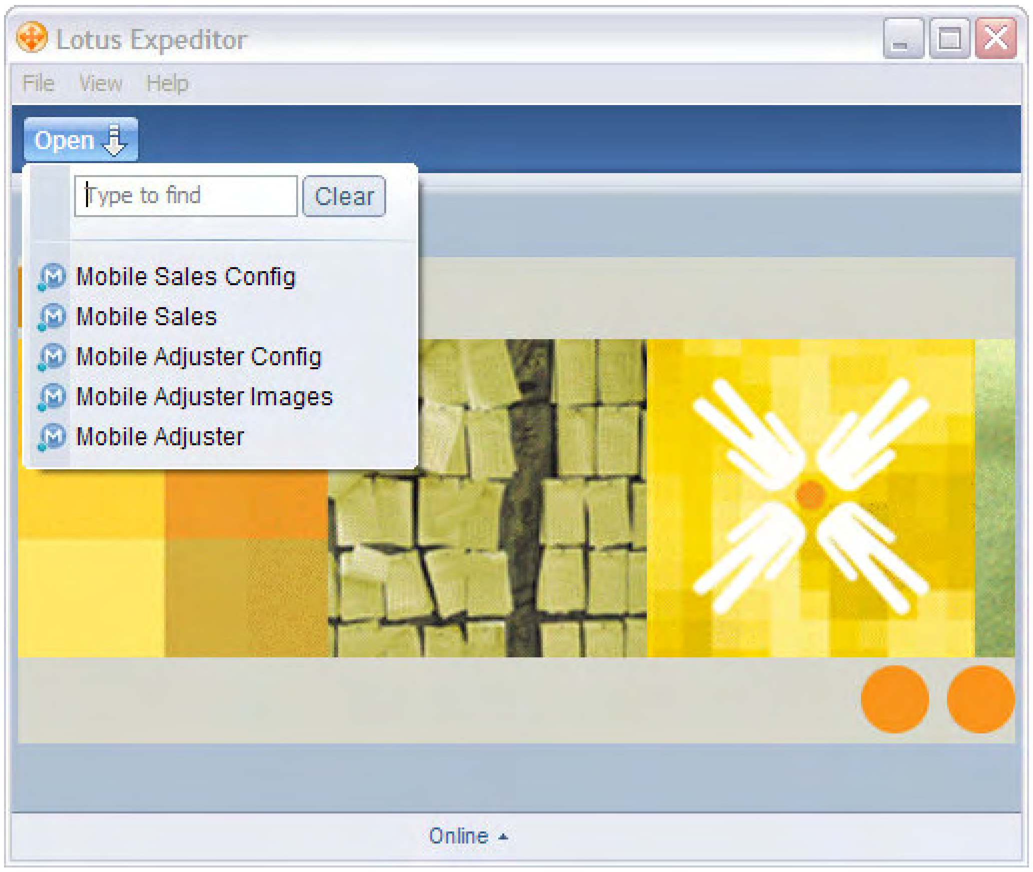


Figure 3: Lotus Expeditor Workbench

The CAI framework is used to assemble, lay out, and wire RCP composite applications without the need for the underlying components to be aware of each other at development time. The components being assembled generally have a GUI, though this is not a requirement. The components also provide a collection of programmatic inputs and outputs, though this is not an absolute requirement. The programmatic outputs are usually fired as the user interacts with the component's GUI. When the programmatic inputs are activated, the GUI is generally updated to reflect the inputs. The programmatic output of a component is declaratively wired to the programmatic input of one or more other components in the RCP composite application, thus allowing the components to communicate without prior knowledge of each other.

Similar to web services, the programmatic inputs and outputs of a component in IBM Lotus Expeditor are described using WSDL. The programmatic inputs can be simple types or complex types defined using XML Schema [8] (XSD). In the current implementation, CAI does not provide a means of declaring the graphical user interface type (e.g. web, SWT, Swing, etc.) of a component. The RCP composite application assembler must have previous knowledge of this if they are restricted in the types of GUI technologies they can use. For example, if the deployment platform does not provide support for portlet interfaces, then an assembler must know which components are built from portlets and specifically avoid those when assembling the RCP composite application.

The layout and wiring of an RCP composite application is executed in a tool called the Composite Application Editor (CAE) by a business analyst or an application assembler. With CAE, the assembler does not need to know how a component was built in order to use it. The desired components can simply be dragged and dropped to add them to an RCP composite application. Once the assembler has decided on a set of components to be included in the RCP composite application, the tool provides a simple method of connecting programmatic inputs to outputs across the application. The adding, removing, and wiring can be done in an iterative fashion to allow the assembler to refine the RCP composite application.

However, finding and using compatible and complementary components from a catalog of existing components is still a problem for the assembler, though this thesis describes methods to simplify that process. Figure 4 shows an RCP composite application with two components. The CityState Picker component (labeled “City View” below) provides a single output, labeled cityState. The HotSpot Finder component provides a single input named SetLocationCityState. The dotted line indicates that the cityState output has been linked to the SetLocationCityState input. Therefore, when the output cityState is fired, the argument of that output will be sent as the argument to the SetLocationCityState input. So even though these two components could have been developed independently, they can be wired together using the CAE tool.

CAE processes the WSDL file that is provided by the component developer to display the available inputs and outputs of a component. WSDL elements, such as message, portType, and binding, together describe the programmatic inputs and outputs of the component. This is advantageous because, as will be shown, we can apply web services matching techniques to find compatible components during the assembly of RCP composite applications.

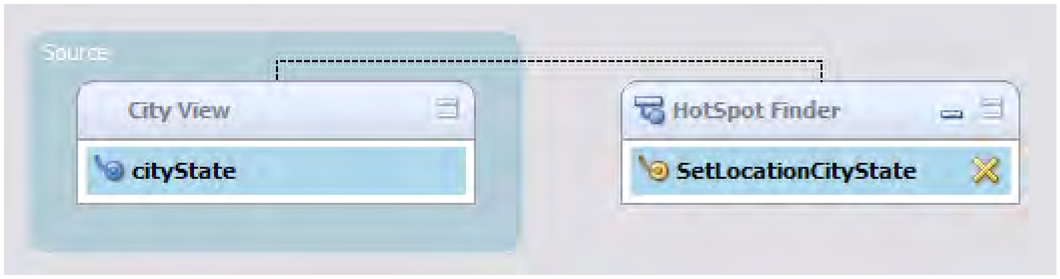


Figure 4: Wiring of two components as shown in the composite application editor

1.4 Component Catalogs

In order to assemble a composite application, the user must have a list of available components from which to select. The standard locations that can be searched are the locally installed system and external component catalogs. In the case of Lotus Expeditor, one of the preconfigured external catalogs is IBM WebSphere Portal. By connecting to WebSphere Portal, the assembler can select from any of the portlets that are installed. By default, WebSphere Portal provides nearly two hundred portlets from which to select. As administrators begin to install more portlets onto the system, the number of selections can increase dramatically. This proves to be a large problem for the assembler, but one we will show how to help solve. There are hundreds of components available, but the only information an assembler may have about them is a title and a short description. By combining the techniques of the semantic web with existing web service matching logic, we will show that the number of available selections can be matched against the existing application and ranked for the assembler. This ranking can simplify the process of finding compatible and complimentary components for the RCP composite application being assembled.

1.5 Thesis Contributions

The main contributions of this thesis are as follows. First it is shown that existing techniques, technologies, and algorithms used for finding and matching web service components (WSDL-based) can be reused, with only minor changes, for the purpose of finding compatible and complementary non-web service based components for RCP composite applications. These components may include graphical user interfaces, which are not an artifact in web service components. By building on the techniques initially developed for web services matching, the problems associated with finding useful and valid components for RCP composite applications using high level concepts are possible. This enables the progressive construction of RCP composite applications from a catalog of available components without deep knowledge of the components in the catalog. Second, it will be shown that the additional characteristics of non-web service based components, specifically graphical user interface details, can be categorized, described, and matched in a similar fashion to the programmatic inputs of the components. Though similar in some respects to web services, these additional characteristics of a component allow for additional match processing logic to be used to provide better results for the novice user when searching for components to integrate into the RCP composite application. Finally, it will be shown using sample applications and scenarios that by taking into account the unique characteristics of an RCP composite application (i.e. coexistence of GUI components), new techniques of merging descriptions can be leveraged to provide better results when searching for new components to add to the RCP composite application.

The structure of this thesis is as follows. Chapter II provides background information on the technologies used in this thesis along with information about related and alternative technologies. Chapter III details the primary contributions of this thesis, specifically the concepts of RCP composite application matching, merging components into a single descriptive format, and modeling of other component characteristics. Chapter IV describes the sample applications that are used to show the viability of the techniques described in this thesis. Chapter V provides a set of experiments, results, and analysis using the sample application components in conjunction with the described techniques. Chapter VI provides a summary of the thesis along with additional information related to the feasibility of the techniques, possible alternatives, and future work in this area.

Chapter II

BACKGROUND

2.1 The Semantic Web

The definition of the semantic web, as given by the World Wide Web Consortium, is "a common framework that allows data to be shared and reused across application, enterprise, and community boundaries" [9]. In practice, this means enhancing software resources with metadata that describes the software in more detail. In order to be useful, the metadata must be in a format that is standardized and machine readable. The two most common formats for the metadata are Resource Description Framework (RDF) [10] and Web Ontology Language (OWL) [11]. Both of these formats provide a method of describing software processes, data types, relationships, and restrictions. There are various arguments for using one format or the other, though those reasons are not important in the context of this thesis. The goal of the semantic web is to describe software resources and data in such a manner that different combinations of software can be assembled automatically based on the characteristics defined in the associated RDF or OWL model.

2.2 Web Services

The primary purpose of a web service is to expose certain pieces of application logic that might be useful in other applications as independent components. A simple example is a currency exchange web service. Such a web service takes a numeric value, an input currency, and an output currency. When the web service is called with these three valid parameters, it returns the equivalent value of the input currency in the output currency. A more complicated web service may be a credit card processing interface that takes an account number, a total cost, and perhaps several other pieces of information in order to create a charge against a credit card. This technology has proven to be very valuable in creating applications that make use of these basic services and do not require each application owner to implement and manage these functions.

As the number and types of web services have expanded, methods have been created to provide developers with better means to search for web services that can be used in their applications. Universal Description, Discovery, and Integration (UDDI) registries often provide a simple search interface that allows a developer to enter keywords and find matches in the text of the WSDL files in the registry. This method has expanded to the creation of a search engine for web services called Woogle [12]. The search engine uses various algorithms that find similar web services operations and sets of operations that can be composed to meet the search criteria. While useful, these methods rely on the WSDL containing similar terms to the given search criteria.

2.3 Semantic Web Service Matching

One of the first and most successful uses of semantic web concepts is being used in web services. Because web services already provide a standard metadata file to describe their programmatic interfaces, adopting that metadata model to incorporate semantic web concepts has not been difficult. The Semantic Annotations for WSDL Working Group [13] at the W3C has developed a standard called the Semantic Annotations for WSDL and XML Schema [14] (SAWSDL). This standard defines a method of including semantic model information into a WSDL document [15]. The semantic model is used to describe all or part of a particular web service using a modeling language. In order to be useful, the modeling language needs to be machine readable and describe the concepts of the web service in ways that can be reasoned about and compared to other models. The model may be added directly to the WSDL, or the WSDL may contain a URL reference to the model (see Figure 5). The standard does not prescribe any particular modeling language, though RDF and OWL are common choices. As an example, a WSDL that describes an XML Schema type named "OrderRequest" could be annotated to reference an "OrderRequest" class in a "purchaseorder" model.

```
<wsdl:types>
  <xs:element name="OrderRequest"
    sawsdl:modelReference=
    "http://www.w3.org/2002/ws/sawsdl/spec/ontology/
        purchaseorder#OrderRequest">
  </xs:element>
</wsdl:types>
```

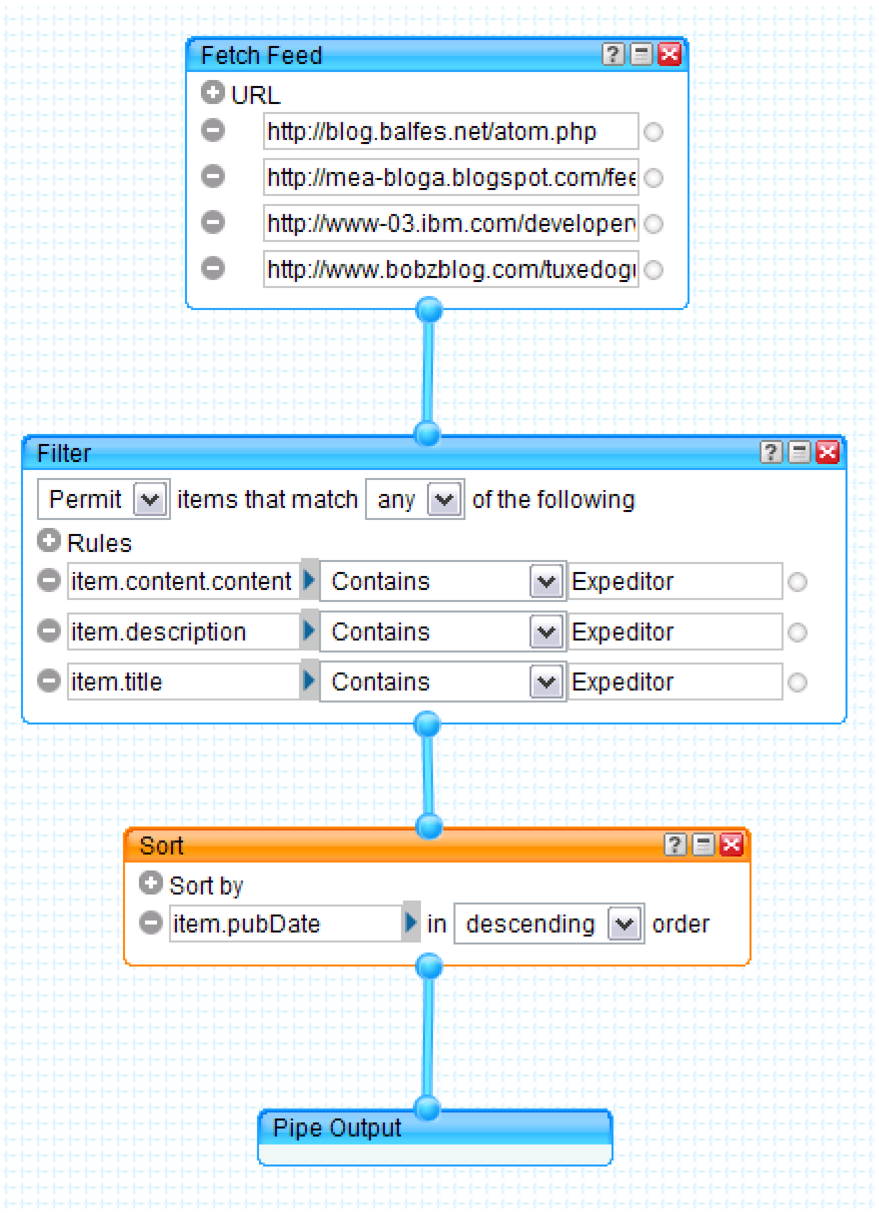
Figure 5: OrderRequest element with semantic model reference

Because the standard does not define any specific language for the model, the "purchaseorder" model can be in any format. The specification does assume that the format used for the model is something that the requester can process.

By enhancing the WSDL with these semantic annotations, the process of searching and finding web services for use with existing applications is improved and better results are provided. No longer does the search have to rely on similar terms and syntax being used in the WSDL file and the search criteria, but the semantic annotations can be used to improve the matching between the inputs and outputs of the web services being considered. In fact, it has been shown [16] that using the two search methods together provides better results when looking for matching web services.

2.4 Other Composite Application Frameworks

In addition to the RCP composite application technologies that are mentioned in this thesis, there are other alternatives available. Two of those include Yahoo! Pipes [17] and the various "Mashup" products, such as Intel Mash Maker [18] and Mash-o-matic [19]. While interesting in their own respects, there are limitations in these technologies that are solved through the use of RCP composite applications running in Lotus Expeditor. Yahoo! Pipes provides a web-based means of pulling data from various data sources, merging and filtering the content of those sources, transforming the content, and finally outputting the content for users to view or for use as input to other pipes. Figure 6 shows an example of a Yahoo! Pipe that pulls data from four RSS and atom feeds, merges the feeds together, filters the feeds based on certain keywords, and finally outputs the results to an HTML page. While certainly interesting and useful, there are several limitations in this platform. The first is the limited set of inputs and outputs. There is no way to use arbitrary inputs or outputs when using this application. There are a limited number of input types and output types from which the user can select. A component in an RCP composite application should be able to accept many different types of inputs and provide many different types of outputs. Secondly, the flow of a pipe is static and sequential. While a user can configure many different inputs, all of the connections are executed in a sequential manner until the single output is reached. With RCP composite applications, the different components in the application can communicate with each other in any manner that the assembler chooses. Finally, Yahoo! Pipes is a server-based technology that makes use of only a web user interface. There is no way for a user to construct and execute a pipe without a network connection and execute the pipe using locally stored data. A pipe can be accessed programmatically, like a web service, but in order to execute the pipe the user must be able to connect to the Yahoo! Pipes server. These same limitations exist in other web portal type solutions such as iGoogle and My Yahoo.



Fetch Feed

URL

http://blog.balfes.net/atom.php

http://mea-bloga.blogspot.com/fee

http://www-03.ibm.com/developen

http://www.bobzblog.com/tuxedogı

Filter

Permit items that match any of the following

Rules

item.content.content Contains Expeditor

item.description Contains Expeditor

item.title Contains Expeditor

Sort

Sort by

item.pubDate in descending order

Pipe Output

Figure 6: Yahoo! Pipe for Lotus Expeditor

Products like Mash-o-matic and the Intel Mash Maker have a similar set of limitations. These products provide a method of taking arbitrary web pages and combining them for use in a single browser view. This imposes the limitation that only web-based technologies can be used. Also, since the data are being pulled from multiple websites, you must be connected to the web and all of the associated websites must be up and functional. These tools also provide means of pulling data from one web page and using it in conjunction with another page that is in the mashup. However, the technique usually used for this is screen scraping. Using this tool, the assembler selects pieces of data from the pages and creates outputs that can be fed to sections of other web pages. While useful, the technique generally uses the Document Object Model (DOM) of the page in conjunction with XPath [20] expressions. These techniques will work, as long as the structure of the web page does not change. For example, if a web page is being used to include stock prices in a mashup and the structure of the stock quote page were to change, it could mean that the function would stop working. Even worse, if the function kept working but was slightly off, stock prices might still be displayed but perhaps associated with the wrong company names. Because there is no concept of API between the provider and the users, as there is in the case of WSDL, these types of applications are prone to breakage and the long-term value remains to be proven [21].

RCP composite applications in Lotus Expeditor solve these problems by providing a method of assembling and executing applications using local application logic. Data can be retrieved from either local or remote data sources. GUIs can be built from a variety of technologies, including portlets, SWT, AWT, Native interfaces, etc. The component author decides on the programmatic inputs and outputs that are exposed by the component and specifies these in a specification (e.g. WSDL), similar to declaring API using public Classes and Interfaces in Java.

Chapter III

COMPOSITE APPLICATION MATCHING

3.1 Existing Techniques for Assembling Composite Applications

Searching for and finding compatible and complementary components for a non-web service based RCP composite application is a difficult problem. Unlike web services, there is currently no standard way of categorizing and cataloging components for use in an RCP composite application. Rather, components are discovered by assemblers who must hunt around the web, in documentation, and searching the locally installed system. This does not provide an easy and manageable means of finding and selecting components. In a portal environment, such as IBM WebSphere Portal, it is possible to query the system for the available components, though the list is returned based on criteria that have no relevance to the application assembler (e.g. alphabetical or last update time). As with large UDDI installations, a large repository of components, such as in a Portal server, can be difficult to search using only text-based search techniques and keywords specified in the components. Interfaces like Google Code Search [22] allow the developer to search application code, but it does not allow you to search using the higher level concepts of a component or a model. On the other end of the problem, having to manually classify and describe every aspect of components for browsing and searching can be a painstaking task when handling a large number of components. However, semantic web techniques have been used successfully for the modeling and matching of web services. The key question we want to address in this thesis is whether we can also leverage semantic web techniques for finding and matching appropriate components for use in Lotus Expeditor composite applications.

After suitable components have been discovered, the **assembly** of RCP composite applications should not require tedious and detailed programming as required of a typical software developer. End users, at least the savvier end users, should be able to compose RCP composite applications with minimal training. For a call center in an enterprise, this may simply mean being able to assemble an RCP composite application on the fly that takes a caller's information in one window and has the input reflected in other components that are used in the call center. For the savvy end user at home or in a small business, this may mean creating a routing application together with the list of errands or deliveries for the day and producing a more optimized route. However, without the ability to find a mapping component that can consume the available input, it could be very difficult for end users to assemble an RCP composite application.

In addition to programmatic inputs, components generally have a graphical user interface. These user interfaces can be built from different technologies, have different deployment requirements, and be usable only in certain human languages. In order to find components for users of RCP composite applications, these additional characteristics may need to be taken into account. For example, if the deployment platform does not support GUIs built using Swing, then Swing-based components should not be given in the search results, or they should be classified lower in the results. It will be shown that these GUI characteristics can be represented using web service and semantic web techniques. When used in combination with the existing web services matching logic, component search capabilities are enhanced. This results in better choices of components for the user in RCP composite applications.

3.2 Modeling Rich Client Components

In Chapter II we described how semantic annotations are used in conjunction with web services to provide developers additional tools to find compatible web services from a collection of available web services. In this thesis we demonstrate that similar techniques, technologies, and algorithms can be leveraged to provide similar function and support for RCP composite application construction. The programmatic inputs and outputs of a component defined in WSDL can be further modeled using semantic web techniques. Additionally, the implementation technology (e.g. SWT, Swing, etc) used to create the graphical user interface can be modeled to further describe the platforms that the component can support.

Most components used in RCP composite applications provide a collection of programmatic inputs and/or outputs, though this is not strictly required. For example, a user profile component may display a user's contact information stored in a Customer Relationship Management (CRM) system. Unlike a web service, a component for an RCP composite application may only take programmatic inputs or only provide programmatic outputs or neither. Similar to web services, the programmatic inputs and outputs of components in Lotus Expeditor are defined using WSDL. Because these components make use of standard WSDL, components for use in RCP composite applications can be annotated to include the additional metadata necessary for them to be matched using existing web services matching technologies and algorithms. One such set of algorithms for web services matching is described by T. Syeda-Mahmood [16]. This work describes combining the use of semantic and ontological matching for the purposes of matching web services. (It should be noted that the use of the term "semantic" [16] refers to text based matching. In this thesis that term is used to mean matching based on semantic markup, such as that defined by the W3C Semantic Web activity [9]). In fact, the matching code that was used to produce the results shown by T. Syeda-Mahmood is the same matching code that is used to conduct the experiments in this thesis. In brief, the matching algorithm works as follows:

1. Using a single input WSDL and a collection of target WSDLs, a score is calculated based on the number of matching terms found between the input WSDL and each target WSDL. A thesaurus, in this case WordNet [23], is used to expand the matching to include synonyms. Thus, this phase is focused on keyword matching.
2. A second search is then invoked using the same input and targets as above. This time the semantic annotations specified in the WSDL files are considered. The semantic models for each component are compared using a custom ontology matching algorithm. This algorithm takes into account the relationships between the elements given, such as inheritance, hasPart, hasProperty, etc. A score for each combination is calculated based on the number of attributes that are matched for each combination.
3. The final score is calculated using a winner-takes-all approach. The maximum of the first score and the second score is reported as the overall matching score for each input and target combination.

Even though we can reuse much of the same logic and algorithms, there are a few fundamental differences when dealing with non-web services based composite applications. In many cases, when searching for a web service, the developer is looking for APIs that can either:

1. Match – Using the output from a single web service and finding a second web service that can take that as input. The developer can continue this process and string together several web services in order to complete a business process.
2. Compose – Starting with a known output and a known input, the developer uses search techniques that can allow them to find one or more services that will transform the output of the first web service into something that can be consumed by the final web service.

The difference with respect to RCP composite applications is that in most cases the goal is not to put together a single business process or tightly link fragments of software processes; rather, the goal is to integrate separately created components together "on the glass" [24] and provide the ability for those applications to communicate or interact without prior knowledge of each other. This means that an RCP composite application may bring together a human resources vacation planning component with a project management component. By linking the two applications together, the project management component could potentially use vacation data in the vacation planning component to adjust project schedules. In no way, however, does the process of scheduling vacation need to be modified in order to use the data. Additionally, the developer of each of these components does not need to have knowledge of the implementation of the other component or the programmatic interfaces.

The markup required for RCP composite application matching is similar to the markup required for web services, at least with respect to data inputs and outputs. A simple example is a CityState Picker component shown in Figure 7. In this component a user can select a state from a list of states and then a city from a list of cities in that state.

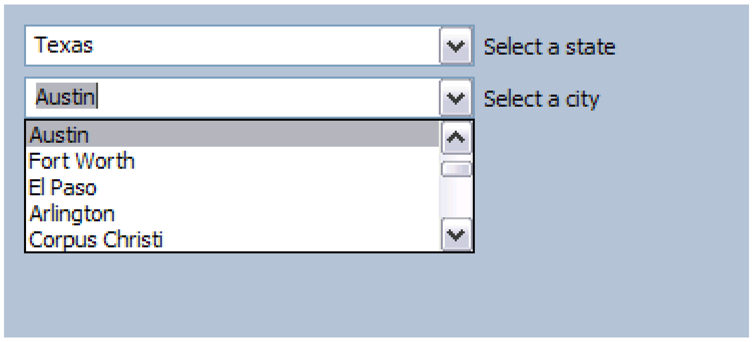
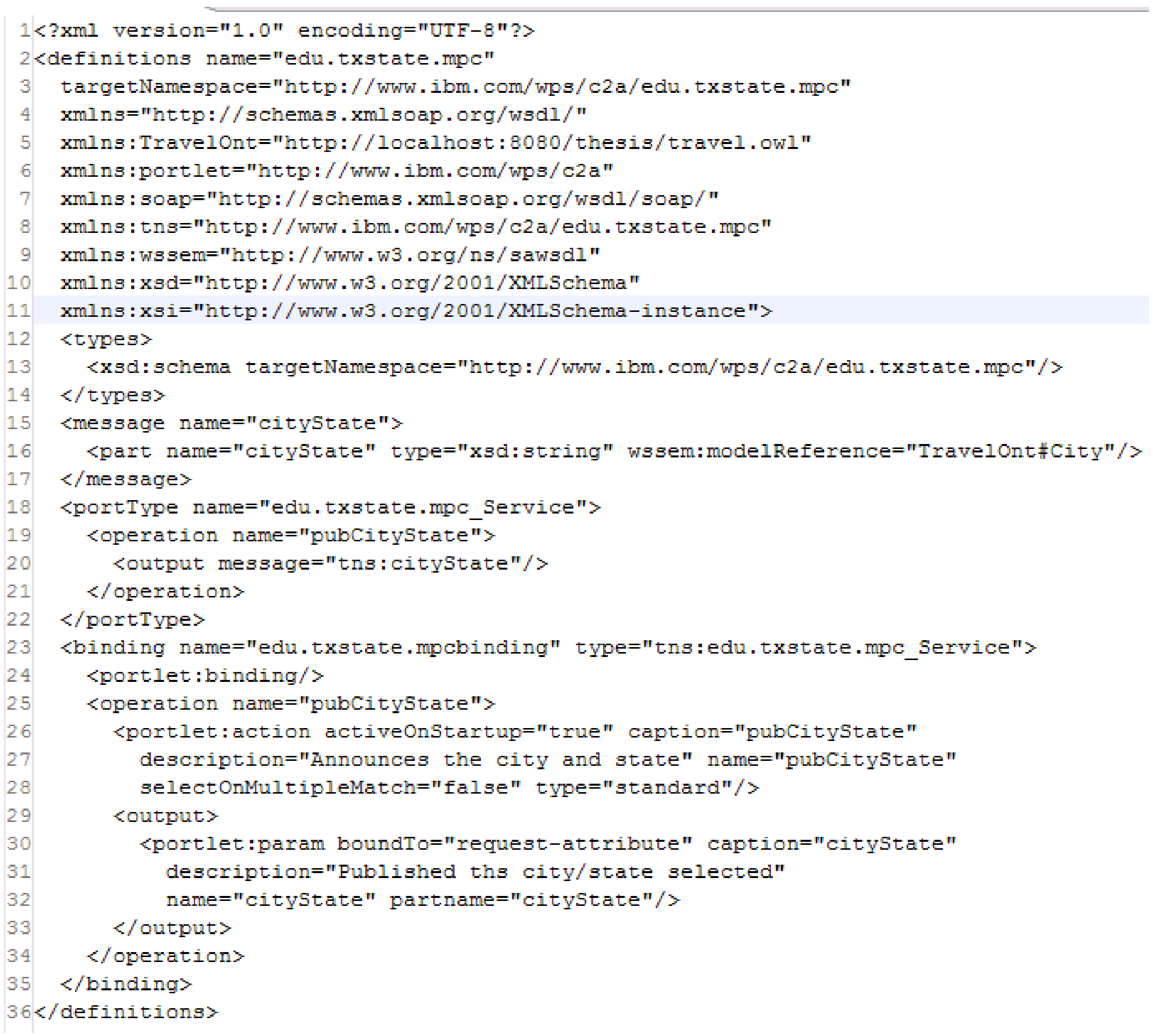


Figure 7: CityState Picker component

When the city is selected, the component outputs (or fires) the city and state that the user selected via the PropertyBroker interface of Lotus Expeditor. The PropertyBroker is responsible for delivering messages between components that have been wired together using the CAE tool. For this component, the state and city are output as a single string encoded using JavaScript Object Notation (JSON). The output definition, including the semantic annotations for this component, is shown in Figure 8.



```
 1<?xml version="1.0" encoding="UTF-8"?>
 2<definitions name="edu.txstate.mpc"
 3  targetNamespace="http://www.ibm.com/wps/c2a/edu.txstate.mpc"
 4  xmlns="http://schemas.xmlsoap.org/wsdl/"
 5  xmlns:TravelOnt="http://localhost:8080/thesis/travel.owl"
 6  xmlns:portlet="http://www.ibm.com/wps/c2a"
 7  xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
 8  xmlns:tns="http://www.ibm.com/wps/c2a/edu.txstate.mpc"
 9  xmlns:wssem="http://www.w3.org/ns/sawsdl"
10  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
11  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
12  <types>
13    <xsd:schema targetNamespace="http://www.ibm.com/wps/c2a/edu.txstate.mpc"/>
14  </types>
15  <message name="cityState">
16    <part name="cityState" type="xsd:string" wssem:modelReference="TravelOnt#City"/>
17  </message>
18  <portType name="edu.txstate.mpc_Service">
19    <operation name="pubCityState">
20      <output message="tns:cityState"/>
21    </operation>
22  </portType>
23  <binding name="edu.txstate.mpcbinding" type="tns:edu.txstate.mpc_Service">
24    <portlet:binding/>
25    <operation name="pubCityState">
26      <portlet:action activeOnStartup="true" caption="pubCityState"
27        description="Announces the city and state" name="pubCityState"
28        selectOnMultipleMatch="false" type="standard"/>
29      <output>
30        <portlet:param boundTo="request-attribute" caption="cityState"
31          description="Published ths city/state selected"
32          name="cityState" partname="cityState"/>
33      </output>
34    </operation>
35  </binding>
36</definitions>
```

Figure 8: WSDL with semantic markup for CityState Picker component

Lines 5 and 9 import the necessary namespaces used to add the semantic annotations. Lines 15 – 17 define a new message named "cityState," which defines the name of the JSON string that will be output. On line 16 you will see that this message has been annotated with a reference to an element in a semantic model. This is shown as *wssem:modelReference="TravelOnt#City"* in the WSDL file. With this annotation, we are describing the message in terms of an OWL class in an OWL model. Continuing along the web services methodology, this message is set as an output of the “pubCityState” operation in a portType (lines 18 – 22) and included in a portlet type binding (lines 23 – 35). With this markup using the standard grammar defined by WSDL and SAWSDL, the component’s output can now be matched against other components’ programmatic inputs using existing matching technologies. By including the annotation, the matching engine is able to match based on capabilities of the component as described in the OWL model. For example, assume the “cityState” component were to be compared against another component that contained the element “county.” The text-based matching would not count these as a possible match because the two strings are not equal (i.e. “cityState” != “county”). However, if the “county” element had a semantic annotation of “TravelOnt#County” in its modelReference attribute, the matching logic would be able to compare the model types City and County. If the model described a relationship between a City and a County, perhaps using the hasProperty OWL attribute, it could be determined that a city is in a county and both are part of a state. Thus, the match would score higher because the modeling analysis would show that these two elements are very closely related.

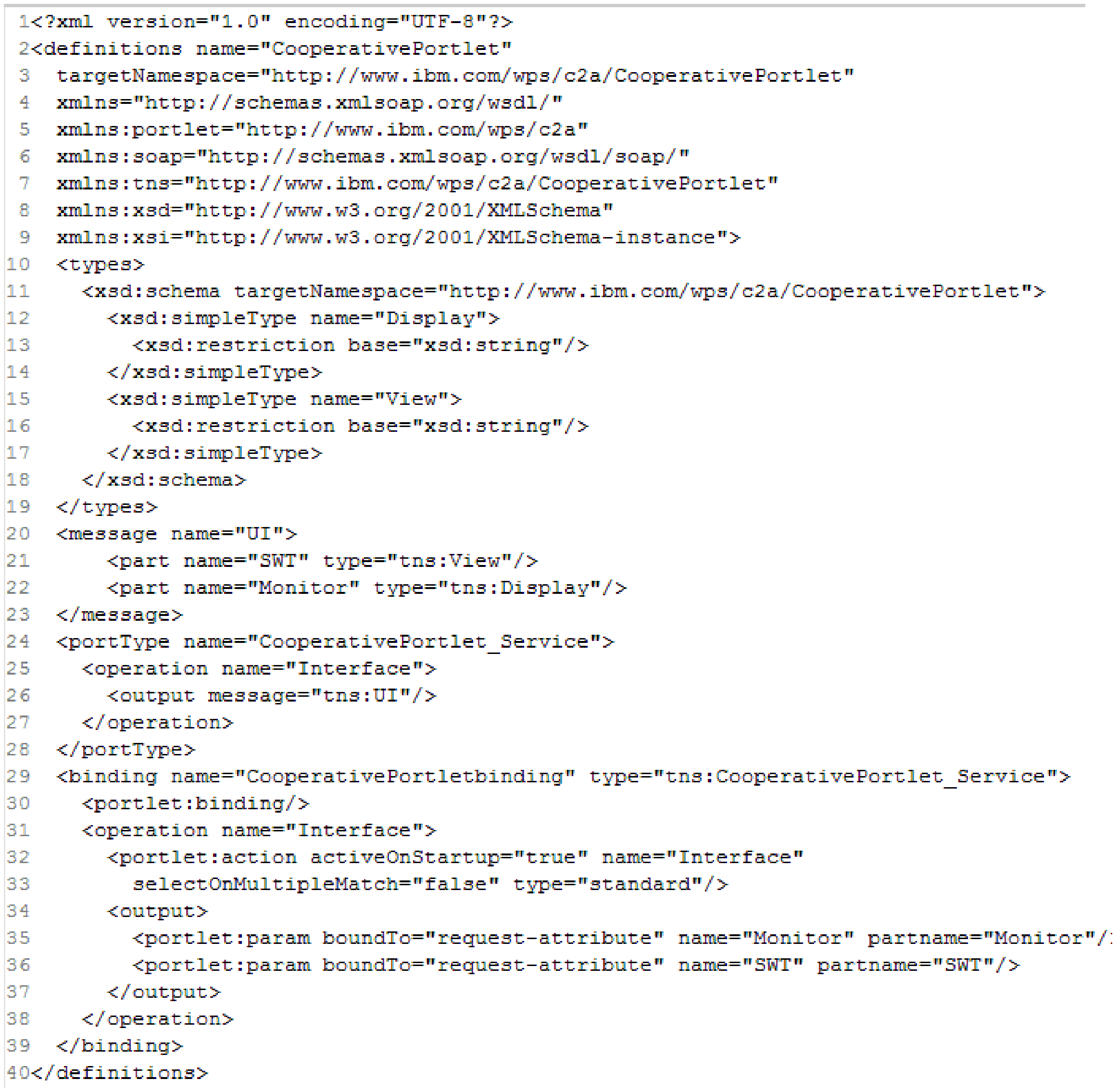
While this example component provides only a basic GUI, more complicated components can be modeled using the same techniques. Components may provide multiple programmatic inputs and programmatic outputs as well as multiple portTypes and bindings. Each of these can cause something different to happen in the component. As such, the messages, portTypes, and bindings can be annotated using any of the available options defined in the SAWSDL specification.

3.3 Modeling Graphical User Interfaces in RCP

Another difference between component and web services is the fact that in general, components are not faceless bits of processing logic. Most components provide some type of GUI, though this is not an absolute requirement. Components that do not provide a graphical user interface may still be useful in a RCP composite application. For example, a currency converter may be useful in an order processing application even though it has no GUI. Prices may be stored in the back-end system in U.S. dollars, but the user could wire the RCP composite application to transform the prices displayed into the local currency of the user.

However, components can have GUIs and it may be necessary to filter or restrict the components found during a search based on the characteristics of the graphical user interface. Such characteristics may include the language the GUI is available in, the technology the GUI is built on, or the complexity of the GUI. For the purposes of this work, two concepts are used to enhance the searching and matching of GUI components: technology used to build the GUI (web, native, Java, etc.) and the recommended display size of the component (projector, desktop monitor, mobile device screen, etc.). The implementation technology used is an important distinction from a technologist point of view because some GUI types, such a GTK+ interface, will not run on a Windows Mobile device. While this type of distinction may not be something an end user thinks about, it is something that will affect them when they attempt to use or build a RCP composite application using incompatible technologies. An obvious characteristic that would be important to an end user is the language in which the component runs. If the RCP composite application is expected to be run in Spanish, a component that only provides a GUI in English may not be usable by the end user. Fortunately, the graphical user interface can be modeled as one or more types of outputs for the component, and the matching technologies can be used to find results that meet the requirements of the RCP composite application. In fact, many characteristics could potentially be modeled using the same technologies, though “graphical user interface” is used in this thesis as a concrete example of one of those characteristics.

Figure 9 shows an example of a WSDL file that is describing the GUI for a given component. In this case, the GUI is being described by two characteristics: Display and View. The Display is described using a combination of type, message, portTypes, and binding. There are two simple types defined, named Display (lines 12 – 14) and View (lines 15 – 17). A message named UI (lines 20 – 23) is then created with two members: an instance of the Display type, named SWT, and an instance of the View type, named Monitor. The portType (lines 23 – 28) marks the UI message as an output with an operation name of Interface. Finally, the Interface operation is added to a binding (lines 29 – 39) so it can be seen by the other members of the RCP composite application.

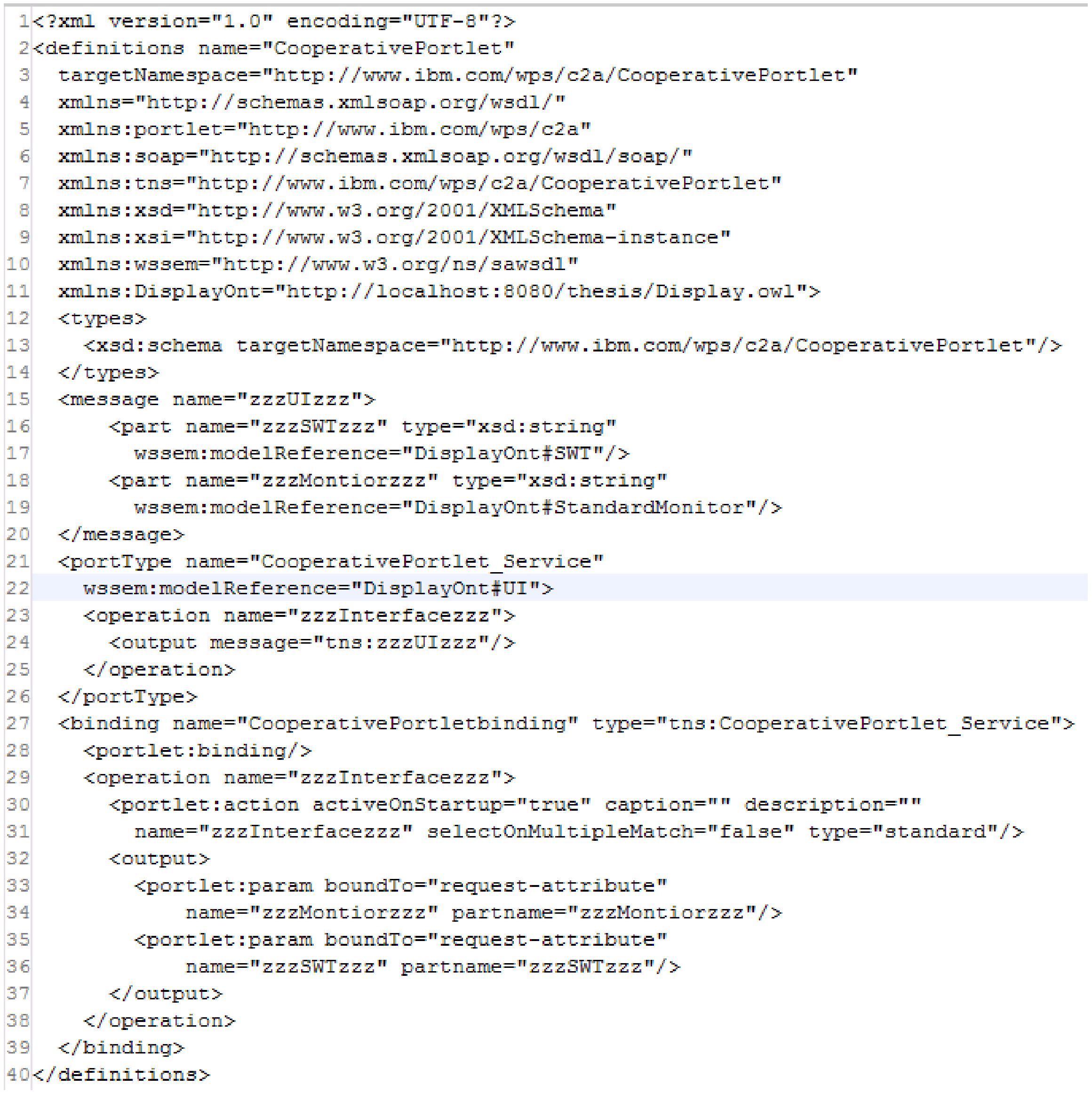


```xml
<?xml version="1.0" encoding="UTF-8"?>
<definitions name="CooperativePortlet"
  targetNamespace="http://www.ibm.com/wps/c2a/CooperativePortlet"
  xmlns="http://schemas.xmlsoap.org/wsdl/"
  xmlns:portlet="http://www.ibm.com/wps/c2a"
  xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
  xmlns:tns="http://www.ibm.com/wps/c2a/CooperativePortlet"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <types>
    <xsd:schema targetNamespace="http://www.ibm.com/wps/c2a/CooperativePortlet">
      <xsd:simpleType name="Display">
        <xsd:restriction base="xsd:string"/>
      </xsd:simpleType>
      <xsd:simpleType name="View">
        <xsd:restriction base="xsd:string"/>
      </xsd:simpleType>
    </xsd:schema>
  </types>
  <message name="UI">
      <part name="SWT" type="tns:View"/>
      <part name="Monitor" type="tns:Display"/>
  </message>
  <portType name="CooperativePortlet_Service">
    <operation name="Interface">
      <output message="tns:UI"/>
    </operation>
  </portType>
  <binding name="CooperativePortletbinding" type="tns:CooperativePortlet_Service">
    <portlet:binding/>
    <operation name="Interface">
      <portlet:action activeOnStartup="true" name="Interface"
        selectOnMultipleMatch="false" type="standard"/>
      <output>
        <portlet:param boundTo="request-attribute" name="Monitor" partname="Monitor"/
        <portlet:param boundTo="request-attribute" name="SWT" partname="SWT"/>
      </output>
    </operation>
  </binding>
</definitions>
```

Figure 9: WSDL describing user interface

If the designer specified that they wanted only components that were written in SWT (the View type) and that worked on a normal computer monitor (the Display type), then this component could be presented to the user as a possible addition to their RCP composite application. However, this matching would only work if we were using a standard vocabulary to describe the interface. Programmers from all over the world may use different terms for the concept of view (e.g. GUI, UI, Interface, UA, etc.). If relying on text-based matching, there would only be moderate success.

Rather than assuming this is the case, we can use some of the techniques described earlier to annotate this WSDL with semantic references. By adopting owl models for GUI and semantic references in the WSDL, we will be able to better reason about what this WSDL is describing. This type of WSDL markup is shown in Figure 10.

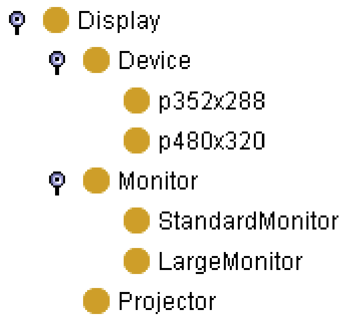


```
<?xml version="1.0" encoding="UTF-8"?>
<definitions name="CooperativePortlet"
  targetNamespace="http://www.ibm.com/wps/c2a/CooperativePortlet"
  xmlns="http://schemas.xmlsoap.org/wsdl/"
  xmlns:portlet="http://www.ibm.com/wps/c2a"
  xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
  xmlns:tns="http://www.ibm.com/wps/c2a/CooperativePortlet"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:wssem="http://www.w3.org/ns/sawsdl"
  xmlns:DisplayOnt="http://localhost:8080/thesis/Display.owl">
  <types>
    <xsd:schema targetNamespace="http://www.ibm.com/wps/c2a/CooperativePortlet"/>
  </types>
  <message name="zzzUIzzz">
      <part name="zzzSWTzzz" type="xsd:string"
        wssem:modelReference="DisplayOnt#SWT"/>
      <part name="zzzMontiorzzz" type="xsd:string"
        wssem:modelReference="DisplayOnt#StandardMonitor"/>
  </message>
  <portType name="CooperativePortlet_Service"
    wssem:modelReference="DisplayOnt#UI">
    <operation name="zzzInterfacezzz">
      <output message="tns:zzzUIzzz"/>
    </operation>
  </portType>
  <binding name="CooperativePortletbinding" type="tns:CooperativePortlet_Service">
    <portlet:binding/>
    <operation name="zzzInterfacezzz">
      <portlet:action activeOnStartup="true" caption="" description=""
        name="zzzInterfacezzz" selectOnMultipleMatch="false" type="standard"/>
      <output>
        <portlet:param boundTo="request-attribute"
            name="zzzMontiorzzz" partname="zzzMontiorzzz"/>
        <portlet:param boundTo="request-attribute"
            name="zzzSWTzzz" partname="zzzSWTzzz"/>
      </output>
    </operation>
  </binding>
</definitions>
```

Figure 10: User Interface WSDL using semantic annotations

In this case, the WSDL has been updated with two new namespaces. Line 10 declares the W3C Semantic Annotations for WSDL namespace and line 11 defines where the Display OWL model can be located, if needed. Note line 10 uses “wssem” for the namespace. This was the original acronym for the W3C proposal that became SAWSDL. It is used here due to dependencies in the matching code being used for the experiments. On lines 17 and 19 you will see the “wssem:modelReference” attribute has been added to the part elements. This is done to indicate that these parts are described by a reference in an external model. Specifically, the part zzzSWTzzz is described by the class SWT in the Display OWL model, and zzzMonitorzzz is described by the StandardMonitor class in the Display OWL model. Finally, on line 22 a model reference to the UI class in the Display OWL model has been added. With this semantic markup we are now able to describe the characteristics of the graphical user interface and display in a non-arbitrary way. Also, as long as consistent or compatible models are used to describe the display, view, and UI concepts, the web services matching code and OWL reasoning engine will be able to find components with appropriate characteristics. You will also notice that many of the names, such as SWT and Display, have had “zzz” added to the name. This is to indicate that the actual name is no longer critical to understand what the WSDL is describing.

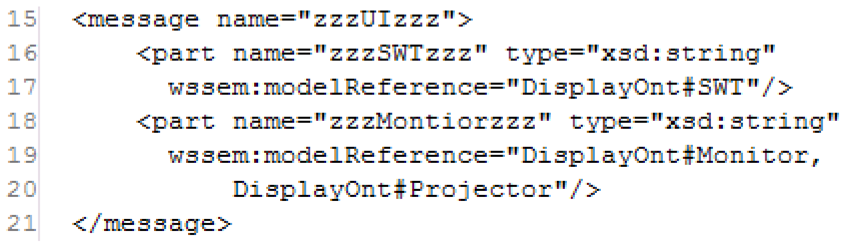
Figure 11 shows the common UI model used in our experiment set. This model describes a single top-level OWL class named Display. There are three subclasses of Display, namely Device, Monitor, and Projector. Further, there are two subclasses of Device, p352x288 and p480x320. For the device case, the two resolutions describe certain types of device interfaces. The p352x288 represents many Windows Mobile smart phones and the p480x320 represents Apple's first generation iPhone. The Monitor class is further subclassed to StandardMonitor and LargeMonitor.



```
Display
    Device
        p352x288
        p480x320
    Monitor
        StandardMonitor
        LargeMonitor
    Projector
```

Figure 11: OWL model for display

Because the classes are not entirely hierarchical, one might assume that you could not describe a view as being appropriate to both a Monitor and a Projector or that it would require two separate GUIs. That is not correct. The SAWSDL spec has solved this problem by allowing multiple model references to be included in an element of a WSDL. So in order to describe a display as supporting both a Monitor and a Projector, it would only be necessary to specify both references, as shown on lines 19 and 20 of Figure 12.



```
15  <message name="zzzUIzzz">
16      <part name="zzzSWTzzz" type="xsd:string"
17        wssem:modelReference="DisplayOnt#SWT"/>
18      <part name="zzzMontiorzzz" type="xsd:string"
19        wssem:modelReference="DisplayOnt#Monitor,
20            DisplayOnt#Projector"/>
21  </message>
```

Figure 12: WSDL message showing multiple model references

Similar to the Display model defined above, a basic ViewType class structure can be defined, as shown in Figure 13. In this case the OWL class list is fairly flat due to the fact that a single component GUI is generally only created from a single type. It is conceivable that additional subclasses could be built. Because Java5 supports Swing and the Abstract Window Toolkit, it would be reasonable to include those as subclasses of the Java5 class.

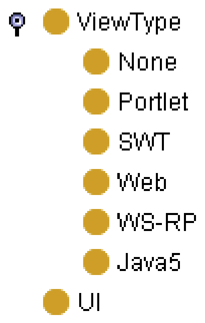


Figure 13: OWL class diagram for ViewType

3.4 Enhancing Web Service's Methods

In the previous section we looked at how components can be modeled using similar techniques to those that are used for web services matching. When used with components applications, the existing web services matching functions can be used without any change. However, there are key differences between an assembly of web services and RCP composite applications. A RCP composite application can include many different components that work together, though not in a predefined order. Users can potentially interact with any member component of the RCP composite application at any point in time. This means that inputs can be received and outputs broadcast at any point in time. This particular characteristic of RCP composite applications works in our favor when looking to find compatible and complementary components. Since the entire UI can be presented to the user at any point in time and the user can interact with any component in the RCP composite application, we can treat the entire RCP composite application as a single component with respect to matching. This allows us to use more advanced techniques during the searching process.

All of the programmatic inputs and outputs from the different components can be merged together prior to a search being executed. Generally, web service matching is done using a single web service WSDL as the input to the matching algorithm. In order to meet the requirement of finding compatible and complementary components that can make use of any of the available programmatic inputs and outputs, the search for components can be structured slightly differently. Rather than using multiple search WSDLs, the components that have already been added to a RCP composite application can be merged together to form a single WSDL to represent the entire set of programmatic inputs and outputs available in that RCP composite application. For example, a RCP composite application that consists of two components with one output each would be represented by a single WSDL that contains both outputs, if they are unique. This can be seen in Figure 14. In the case where the two components share a single output type, only a single instance of that output would be needed in the merged WSDL, though two portType and binding entries may be necessary to indicate the unique actions, as shown in Figure 15.

```
...

<message name="customer_id">
  <part name="customer_id" type="xsd:string"
  wssem:modelReference="OrderOnt#Customer_ID"/>
</message>
<message name="order_id">
  <part name="order_id" type="xsd:string"
  wssem:modelReference="OrderOnt#Order_ID"/>
</message>

...

<binding name="CooperativePortletbinding" type="tns:CooperativePortlet_Service">
  <portlet:binding/>
  <operation name="order.details">
    <portlet:action activeOnStartup="true" caption="order.details"
      description="Get details for specified order id"
      name="order.details" selectOnMultipleMatch="false" type="standard"
      actionNameParameter="ACTION_NAME"/>
    <input>
      <portlet:param boundTo="request-parameter" caption="order_id"
        name="order_id" partname="order_id"/>
    </input>
  </operation>
</binding>
<binding name="CooperativePortletbinding" type="tns:CooperativePortlet_Service">
  <portlet:binding/>
  <operation name="customer.details">
    <portlet:action activeOnStartup="true" caption="customer.details"
      description="Get detail for customer with specified customer id"
      name="customer.details" selectOnMultipleMatch="false" type="standard"
      actionNameParameter="ACTION_NAME"/>
    <input>
      <portlet:param boundTo="request-parameter" caption="customer_id"
        name="customer_id" partname="customer_id"/>
    </input>
  </operation>
</binding>
```

Figure 14: WSDL message with two unique types and actions

```
...

<message name="order_id">
  <part name="order_id" type="xsd:string" wssem:modelReference="OrderOnt#Order_ID">
  </part>
</message>

...

<binding name="CooperativePortletbinding" type="tns:CooperativePortlet_Service">
  <portlet:binding/>
  <operation name="order.details">
    <portlet:action activeOnStartup="true" caption="order.details"
      description="Get details for specified order id"
      name="order.details" selectOnMultipleMatch="false" type="standard"
      actionNameParameter="ACTION_NAME"/>
    <input>
      <portlet:param boundTo="request-parameter" caption="order_id"
        name="order_id" partname="order_id"/>
    </input>
  </operation>
</binding>
<binding name="CooperativePortletbinding" type="tns:CooperativePortlet_Service">
  <portlet:binding/>
  <operation name="SHIPPINGORDERSordersForMonth">
  <portlet:action actionNameParameter="ACTION_NAME" activeOnStartup="true"
     caption="orders.for.month" description="get orders for specified month"
     name="SHIPPINGORDERSordersForMonth" selectOnMultipleMatch="false" type="standard">
    <portlet:constant-params>
      <portlet:constant-param name="defaultMonth" value="January"/>
     </portlet:constant-params>
   </portlet:action>
   <output>
     <portlet:param boundTo="request-parameter" caption="order_id"
        name="order_id" partname="order_id"/>
    </output>
  </operation>
</binding>
```

Figure 15: WSDL message with one common type and two unique actions

While the merging of WSDLs is a good solution, it is not without its own limitations. The matching algorithm being used takes into account two characteristics during matching: the semantic annotations in the “wssem” attributes and the name attributes of the elements. When merging WSDLs, the name that is used can come from any of the input WSDLs, if the elements are determined to be equal. As will be seen in the experimental results, it is possible to get different matching scores depending on which name is used to create the merged WSDL. The results of experiment three in chapter V show that the merging of the WSDLs can provide better results than processing the WSDLs independently. We found that the best way to avoid the problem of different results based on the name selected is to always use semantic annotations. When semantic annotations are used to annotate a component, the name that is used during the search does not affect the score.

Chapter IV

RCP COMPOSITE APPLICATION ASSEMBLY

In order to validate the usefulness of the enhanced RCP composite application matching with semantic web technology, experiments have been conducted using two separate and different RCP composite applications. This chapter will describe in more detail how to create components and annotate them with semantic attributes to allow them to be used in conjunction with the semantic web matching functions described in this thesis. The sample applications selected for the experiments were chosen because they represent two different types of valid RCP composite applications. The travel scenario represents a hybrid RCP composite application that includes Eclipse SWT GUI and a web application accessed via a web browser. This is the type of application that would be constructed when selecting from a list of existing components. The order tracking application represents a composite application that could be deployed to an RCP product like Lotus Expeditor or to a cooperative portal environment like IBM WebSphere Portal or BEA WebLogic Portal. This is the type of application that would be built with the original intent of deploying to an SOA environment.

4.1 Building a Component

In order to create a component for reuse in a RCP composite application, a few additional steps are required beyond what is necessary to create a stand-alone application or component. However, the majority of the process is similar to creating standard applications.

One of the first steps that any developer will need to take when building an application is to decide on the interface technology. This may be Java Swing, Flex, native, portlets, etc. This is no different than when creating components for a composite application. In composite application environments like BEA WebLogic Portal, the only choice for a GUI may be portlets. If the developer is building components for use in composite applications that will be deployed to Lotus Expeditor, the user can select from many technologies, including SWT, Java Swing, native interface, portlet, etc. The next step in building a component for an application is locating the data that is to be presented in the GUI. Often this information consists of records from a relational database or some other back-end data source. Up to this point there is no difference in the process between creating components for an RCP composite application and creating a standard application.

If the developer decides that they would like to expose their newly created components such that they can be reused in an RCP composite application, then a few additional steps must be taken. The first step is to decide which programmatic inputs and outputs should be exposed. In a view that displays a user’s contact information, the developer may choose to publish the user ID of the person displayed in the view. Also, the developer may decide that the component will take an email address as an input. When the input is received, the GUI will be updated to show the contact information for the user associated with the email address. In order to make this functionality work, the developer is required to do a couple of things. First the developer must create a WSDL file that describes the programmatic input and output of the component – in this case one input for email address and one output for user ID. This WSDL is used to notify the composite application framework of the inputs and outputs associated with this component.

The second step is to write a bit of code to deal with the programmatic input and the output. The interface used to programmatically send and receive components is provided by the PropertyBroker component of Lotus Expeditor. (The same interface is available on IBM WebSphere Portal for Portal based composite applications.) Using the PropertyBroker interface, the code can be notified of incoming email addresses. When an email address is received, the code can respond in some way. In this example, a good response might be to update the GUI with the contact information for the user associated with the given email address. Using a similar set of APIs, the developer can also publish the user ID of the person shown in the contacts view. It should be noted that the developer does not require any knowledge of other components that may decide to make use of these inputs and outputs.

At this point, the component is ready for use in an RCP composite application. The assembler then uses the Composite Application Editor (CAE) tool to find and select components for use in the RCP composite application and wire together their programmatic inputs and outputs. Components can be added from an associated Portal based catalog, from the running instance of the Lotus Expeditor Client, or from Eclipse update sites. However, this is where our primary problem lies. With all of these possible repositories and numerous components that can exist in each repository, how is an assembler to know which components to add to their RCP composite application without deep knowledge of the catalogs and the separately developed components. This is where semantic annotations and the matching logic can be used to simplify the process of finding compatible and complementary components for a RCP composite application. Before this can be done though, the components need to be annotated with additional semantics.

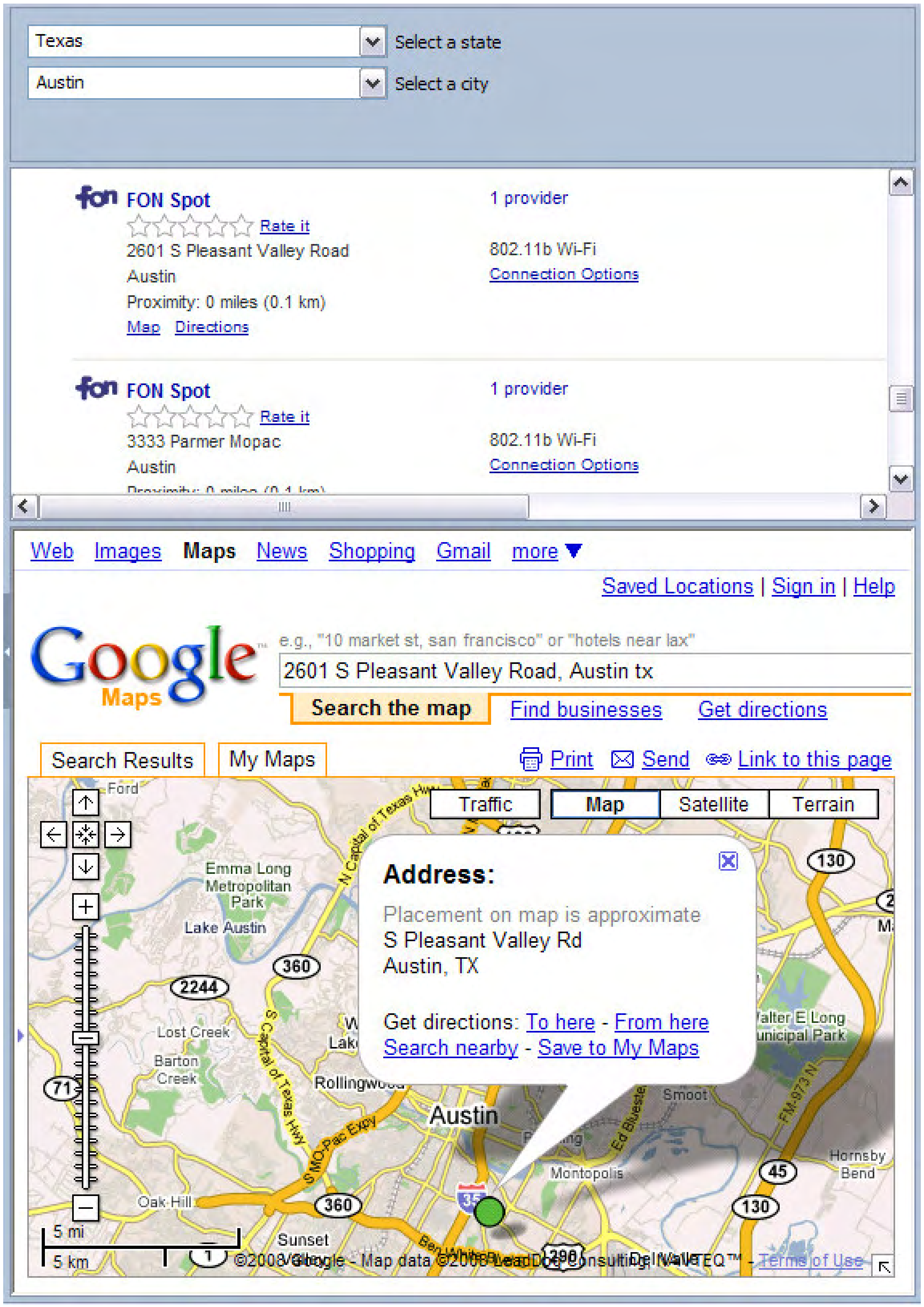
4.2 Annotating Components with Semantic Information

While creating a component for use in a RCP composite application may seem like a long, arduous process, it is fairly straightforward and repeatable among all different types of components. In order to perform semantic matching of components, the developer should add semantic annotations to the WSDL that describe the programmatic inputs and outputs of their component. As discussed in this thesis and other papers on web services matching, the semantic annotations can provide better matching results then straight text-based matching. The developer should therefore update the WSDL with references to semantic models that describe the programmatic inputs and outputs of their component. Additionally, the developer should describe the GUI of their component by using the previously mentioned methods for describing the graphical user interface (see Chapter III). Semantic annotations only work if there is a unified ontology model. Therefore, it may be necessary to create semantic models for the components. Tools such as the Protégé-OWL [25] editor can be used to create OWL-based semantic models if none exist. However, a given enterprise may have a collection of models that already exist to describe the data and processes used in the enterprise. Further, a given industry may have a collection of models already created that describe the unique characteristics of that industry. If the component being built is intended for use in a given enterprise or industry, care should be taken to use existing semantic models where it makes sense. Since the SAWSDL specification allows multiple models to be attached to a given element, it may be appropriate to provide one or more enterprise, industry, and custom models to a particular element in the component.

4.3 Sample Applications

Chapter V will describe how the two sample applications below can be assembled progressively using semantic annotations and adopting semantic web service matching logic to find compatible and complementary components. In what follows, the function of the two chosen applications are discussed.

The travel scenario RCP composite application consists of three separate components. The first component is the CityState Picker described to a small extent previously. The CityState Picker allows a user to first select a state from a select box and then select a city from a second select box. When the city is selected, the component publishes the city and state selected as a JSON encoded string. The second component is a HotSpot Finder component. This component is coded as an instance of the Eclipse SWT Browser, which is programmatically driven to different URLs based on the inputs. In order to provide interesting content, the JiWire [26] website is accessed by the browser. The HotSpot Finder takes as input a city and state encoded as a JSON string. When this input is received, the browser is directed to a URL on the JiWire website, which provides a listing of wireless Internet access points in the given city and state. The HotSpot Finder also provides one output, an address. When a user double-clicks on an address shown in the HotSpot Finder, the address selected with the double-click is published as a JSON encoded string. The third component is also coded using an instance of the Eclipse SWT browser. In this case, the component takes as input an address. Based on this address, the browser loads a map for the address using Google Maps to provide the actual content. This Google Maps component does not provide any outputs. The picture of the complete RCP composite application can be seen in Figure 16.


Texas Select a state
Austin Select a city

FON Spot
Rate it
2601 S Pleasant Valley Road
Austin
Proximity: 0 miles (0.1 km)
Map Directions
1 provider
802.11b Wi-Fi
Connection Options

FON Spot
Rate it
3333 Parmer Mopac
Austin
1 provider
802.11b Wi-Fi
Connection Options

Web Images Maps News Shopping Gmail more
Saved Locations | Sign in | Help
e.g., "10 market st, san francisco" or "hotels near lax"
2601 S Pleasant Valley Road, Austin tx
Search the map Find businesses Get directions
Search Results My Maps Print Send Link to this page
Traffic Map Satellite Terrain

Address:
Placement on map is approximate
S Pleasant Valley Rd
Austin, TX

Get directions: To here - From here
Search nearby - Save to My Maps

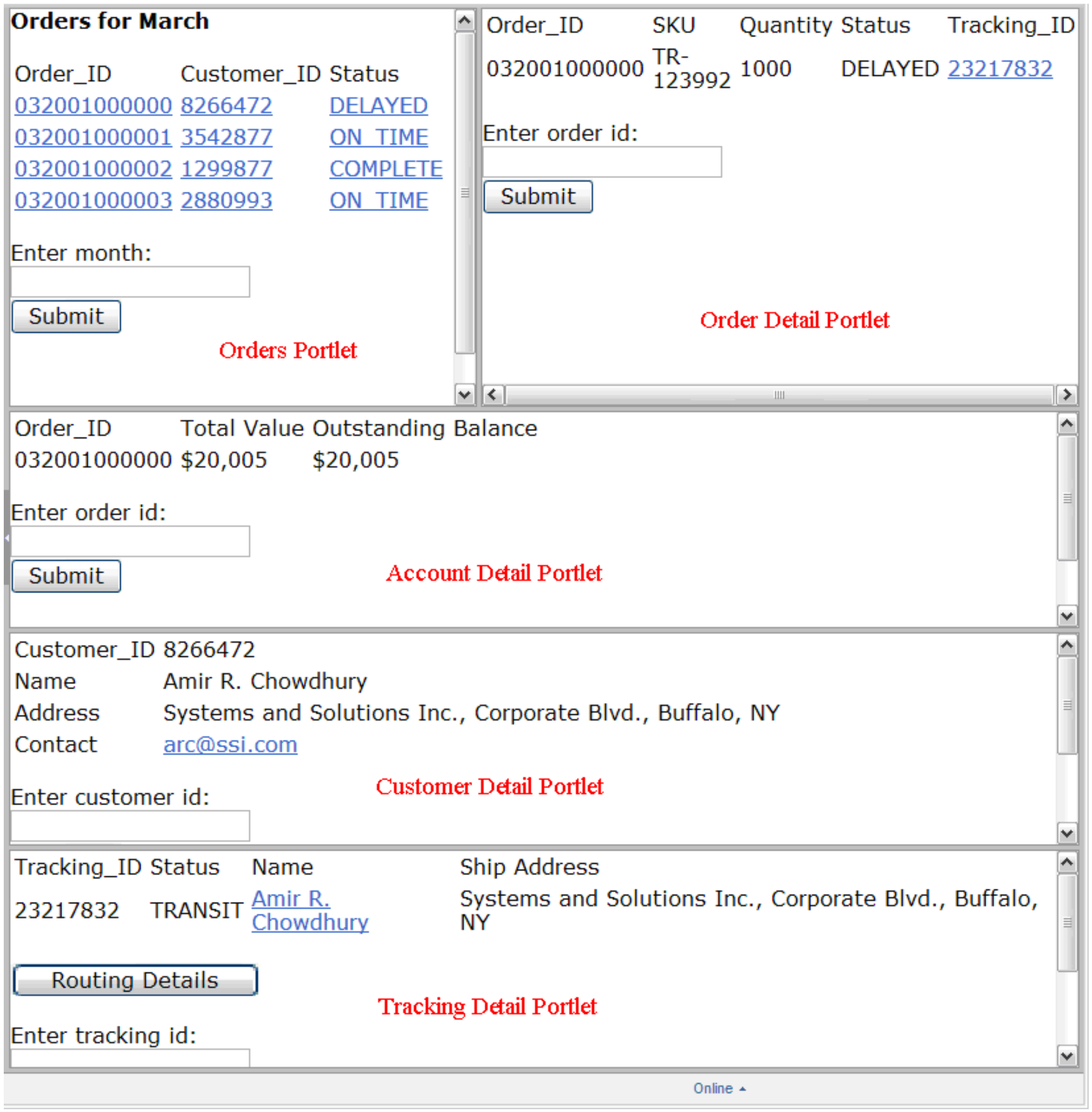
Figure 16: Complete travel scenario application

Each of the WSDLs associated with these components has been created based on the inputs and outputs they define using the Wiring Properties Editor provided with the Lotus Expeditor Toolkit [27]. The WSDLs have been annotated with semantic information using the SAWSDL defined attributes to WSDL. An existing travel related OWL model created by Holger Knublach was used to annotate this new application. The travel model is shown in more detail in "Ontology-Driven Software Development in the Context of the Semantic Web: An Example Scenario with Protégé/OWL" [28]. A single subclass named Address was added to this model as a subclass of destination. This is not absolutely necessary, as the existing ContactAddress class could have been used. This was done only to provide a simplification of the OWL model. The complete WSDLs for the three components can be found in Appendix A.

The second scenario is an order tracking scenario. In this scenario, there are five individual components, with several inputs and outputs. The individual components are built as portlets configured to communicate as part of a RCP composite application. The base code for this scenario was taken from the Cooperative Portlets sample provided as part of Rational Application Developer 7.0. The same sample is described in detail in the article "Developing JSR 168 compliant cooperative portlets" [29]. The code was reused with only minor changes; small errors were corrected in the application code. The WSDLs provided also had many errors that were corrected by creating new WSDLs from scratch. The sample consists of five components with a different collection of inputs and outputs. Those components are as follows:

1. Orders Portlet – Displays a list of existing orders for a specific month. The component will accept a month input and will output a month, an order_id, and a customer_id.
2. Order Detail Portlet – Displays the details of a specific order. The component will accept an order_id as an input and will output a tracking_id.
3. Tracking Detail Portlet – Displays the tracking information related to a specific order. The component will accept a tracking_id as input and will output a customer_name.
4. Customer Detail Portlet – Displays customer information. The component will accept a customer_id and a customer_name as input and does not provide any outputs.
5. Account Detail Portlet – Displays the account details of a particular order. The component will accept an order_id as input and does not provide any outputs.

The completed application can be seen in Figure 17. The layout is arbitrary and can be adjusted in any manner without affecting functionality.



Orders for March

Order_ID	Customer_ID	Status
032001000000	8266472	DELAYED
032001000001	3542877	ON_TIME
032001000002	1299877	COMPLETE
032001000003	2880993	ON_TIME

Enter month:

Order_ID	SKU	Quantity	Status	Tracking_ID
032001000000	TR-123992	1000	DELAYED	23217832

Enter order id:

Order_ID	Total Value	Outstanding Balance
032001000000	$20,005	$20,005

Enter order id:

Customer_ID	8266472
Name	Amir R. Chowdhury
Address	Systems and Solutions Inc., Corporate Blvd., Buffalo, NY
Contact	arc@ssi.com

Enter customer id:

Tracking_ID	Status	Name	Ship Address
23217832	TRANSIT	Amir R. Chowdhury	Systems and Solutions Inc., Corporate Blvd., Buffalo, NY

Enter tracking id:

Figure 17: Completed order tracking application

In order to semantically annotate this application, a new OWL model was created and the WSDLs for each of the components were annotated with the appropriate markup. The new model, named Order.owl, contains classes to represent each of the programmatic inputs and outputs of this application. For a larger order tracking system, a more complex model would be required. The model used for this application can be seen in Figure 18. The complete WSDLs for this application can be found in Appendix B.

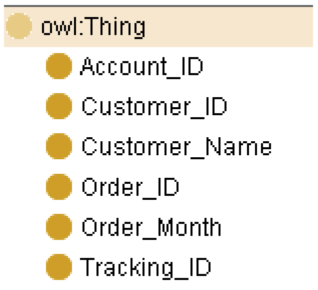


Figure 18: Order owl model

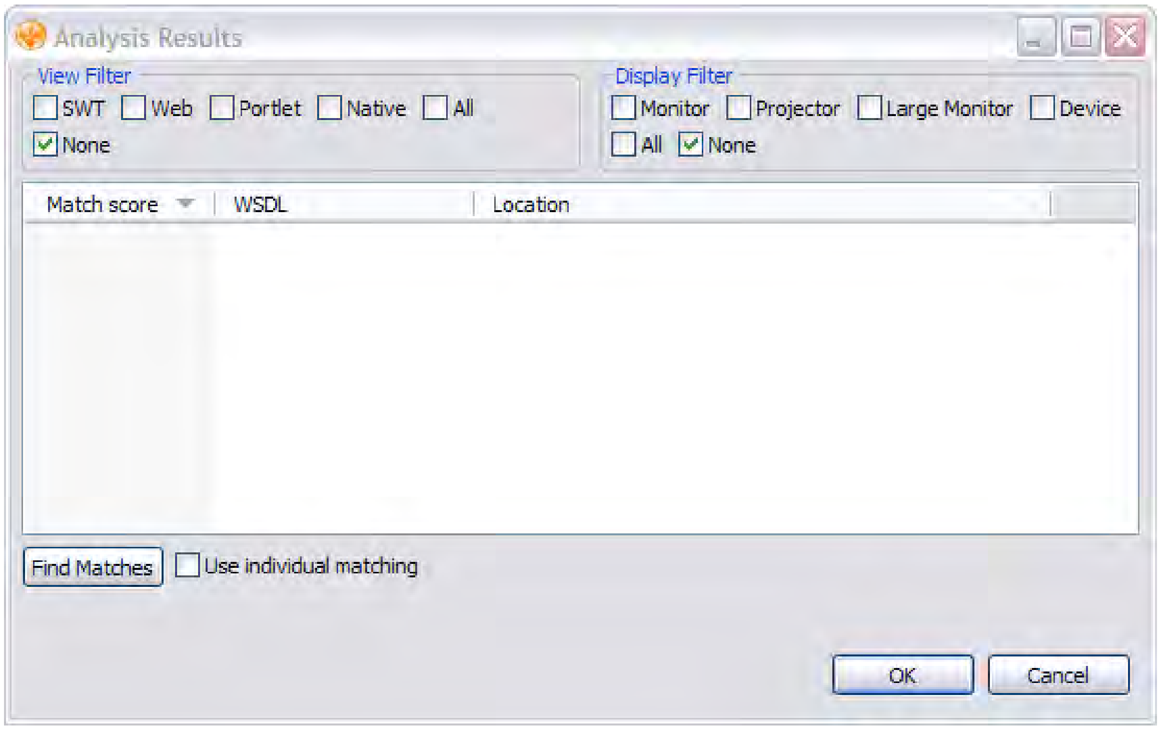
Chapter V

EXPERIMENTAL RESULTS

5.1 Introduction

The experiments described in this section were completed using the sample applications previously described in conjunction with the IBM Lotus Expeditor 6.1.2 release. The experiments were conducted on a Lenovo ThinkPad T60p running Microsoft WindowsXP SP2. An Apache HTTP server in conjunction with an IBM WebSphere Portal Server 6.0 was used to simulate the component library.

In order to drive the test cases and report results, a graphical user interface component was created (see Figure 19). This component reads the currently executing composite application and drives the test cases. The "View Filter" and "Display Filter" sections allow the user to set the graphical user interface search criteria. The main table displays the score associated with each of the target WSDLs that were included in a search request. The "Find Matches" button starts the search process. Finally, the "Use individual matching" checkbox allows the user to specify which type of matching will be used. If checked, each of the components' WSDLs in the current RCP composite application will be matched individually to the target WSDLs in the respository. If not checked, a merged search WSDL will be used in the matching process.



View Filter: SWT [ ] Web [ ] Portlet [ ] Native [ ] All [ ] None [✓]

Display Filter: Monitor [ ] Projector [ ] Large Monitor [ ] Device [ ] All [ ] None [✓]

Use individual matching [ ]

Figure 19: Analysis Results Dialog

In addition to the application components described in Chapter IV, section 4.3, an additional four WSDLs with semantic annotations were included in the target component repository. These WSDLs are named SourceInterface.wsdl, SourceInterfaceV1.wsdl, TargetInterface.wsdl, and TargetInterfaceV1.wsdl. The full listing of these WSDLs can be seen in Appendix D. These additional WSDLs were added to the repository in order to simulate other components of composite application. These WSDLs describe components for a retail order system, which would be a valid RCP composite application, though not applicable as components in two applications being constructed as part of this thesis.

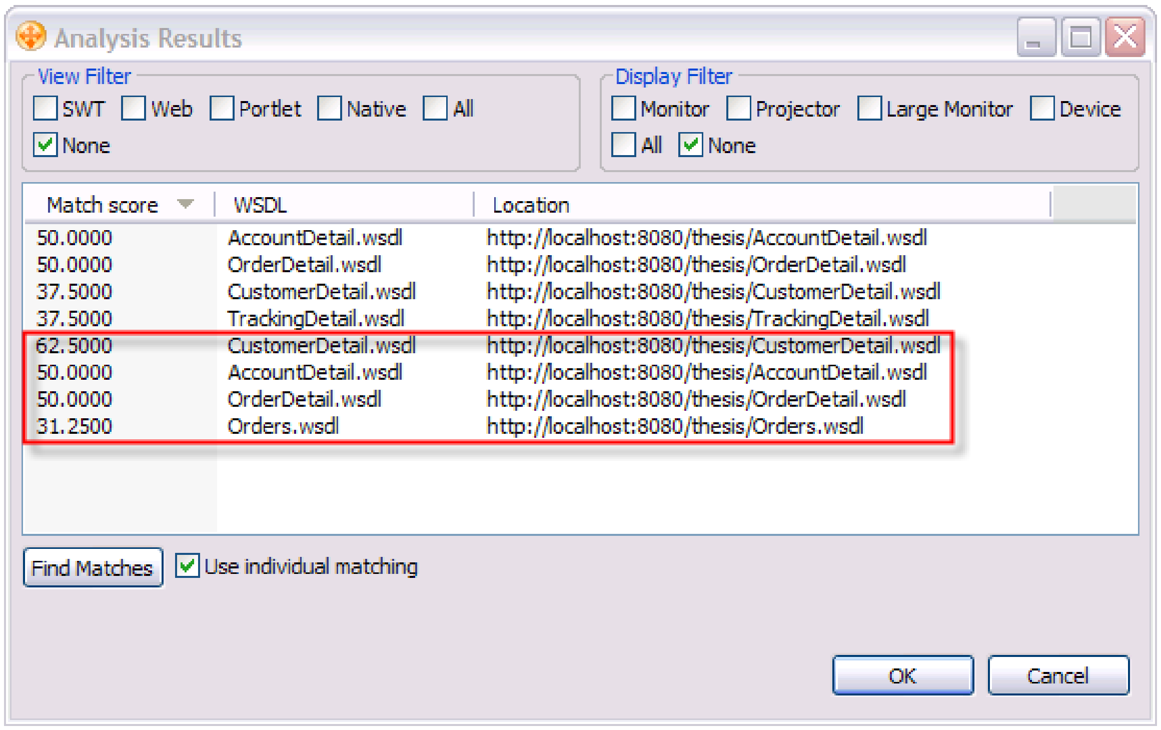
5.2 Experiment One – Basic Matching

The first experiment shows simple matching using two component WSDLs and the matching logic. The input for this scenario is the CityStatePicker.wsdl and the target is the HotSpotFinder.wsdl, both shown in Appendix A. When run through the matching logic, a score of 50 is produced. This is a reasonable score because of the differences in the two WSDLs. The CityStatePicker.wsdl file defines a single message, cityState, and the HotSpotFinder.wsdl defines two messages, city and address. As will be seen later, this is a fairly high score when matching. In order to show the results of the semantic matching only, a modified version of the HotSpotFinder.wsdl is used. In the modified version, the identifying names such as city and address are replaced with non-descriptive strings such as vvv and ddd. Because these do not match fields in the CityStatePicker.wsdl, the ontological score is always returned. The resultant score in this case is 37.50. This lower score can be accounted for based on the fact that only the message elements have models attached to them.

5.3 Experiment Two – Merged WSDL Matching

The second experiment shows the effect of using the merged WSDL process to find compatible components for a RCP composite application. The two inputs for this experiment will be the Orders.wsdl and TrackingDetail.wsdl. These will be matched against the other three WSDLs that are part of the Order Tracking scenario, specifically AccountDetail.wsdl, OrdersDetail.wsdl, and CustomerDetails.wsdl. Given this setup, any of the three target WSDLs could be a good match because each of them have inputs that can be satisfied by the available outputs of the Orders and Tracking Details components.

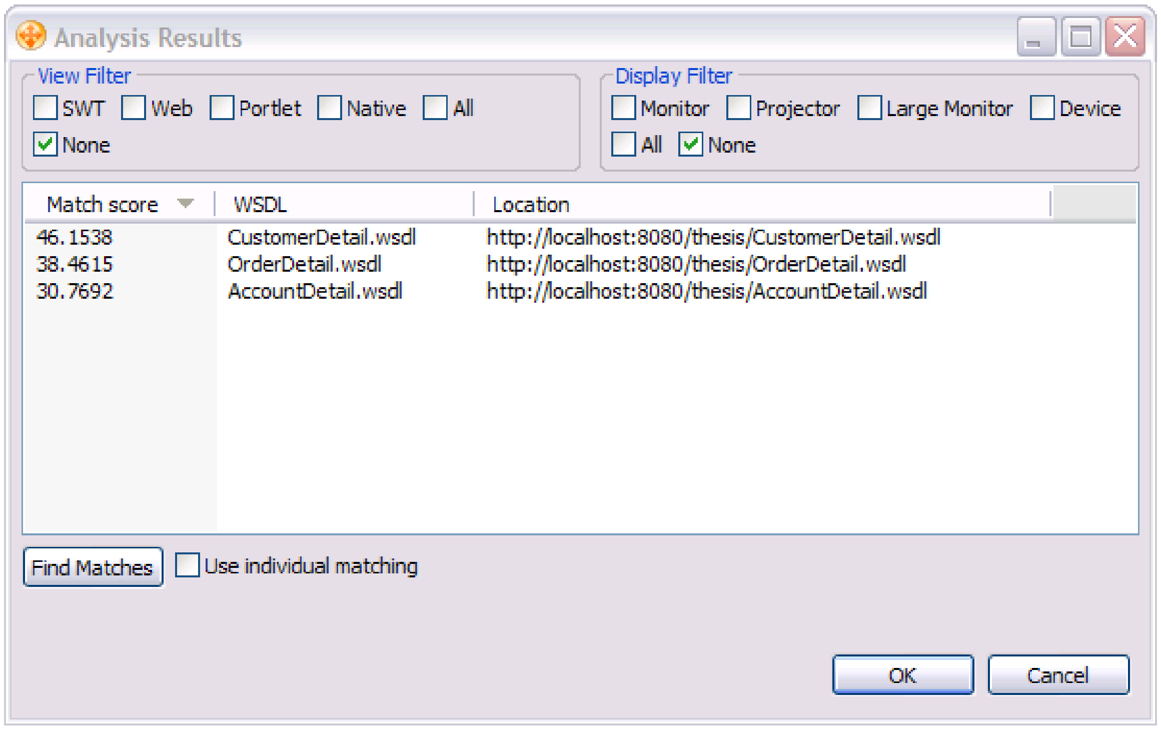
First the two input components are individually matched against the other three. The results are shown in Figure 20. The first four entries in the list are the result of matching against the Orders.wsdl; the second four results, grouped in the box, are the result of matching against the TrackingDetails.wsdl. As you can see, the Customer Detail component has the highest overall match value of the possible choices. This makes sense because the single output of the Tracking Details component matches one of the two inputs to the Customer Details component. When we look at the results for the Orders component, we see the Customer Detail component is ranked lower than either the Account Detail or the Order Detail component and equally scored against the Tracking Detail component. The Account Detail and Order Detail components also scored fairly well in the match against the Tracking Details component, so perhaps those are better choices.



Match score	WSDL	Location
50.0000	AccountDetail.wsdl	http://localhost:8080/thesis/AccountDetail.wsdl
50.0000	OrderDetail.wsdl	http://localhost:8080/thesis/OrderDetail.wsdl
37.5000	CustomerDetail.wsdl	http://localhost:8080/thesis/CustomerDetail.wsdl
37.5000	TrackingDetail.wsdl	http://localhost:8080/thesis/TrackingDetail.wsdl
62.5000	CustomerDetail.wsdl	http://localhost:8080/thesis/CustomerDetail.wsdl
50.0000	AccountDetail.wsdl	http://localhost:8080/thesis/AccountDetail.wsdl
50.0000	OrderDetail.wsdl	http://localhost:8080/thesis/OrderDetail.wsdl
31.2500	Orders.wsdl	http://localhost:8080/thesis/Orders.wsdl

Figure 20: Results of individual matching in experiment 2

Given this ambiguity, how do we decide which component to add to the RCP composite application? This is where the merged WSDL search can assist. If we do a merge WSDL search, we combine the inputs and outputs of the two given components and match those against the remaining components in the catalog. The results of this search are shown in Figure 21. In this case, we can now see that the Customer Detail component is probably the best component to add to the RCP composite application. It has scored the highest value when compared against the complete application.



View Filter: [✓ None]

Display Filter: [✓ None]

Match score	WSDL	Location
46.1538	CustomerDetail.wsdl	http://localhost:8080/thesis/CustomerDetail.wsdl
38.4615	OrderDetail.wsdl	http://localhost:8080/thesis/OrderDetail.wsdl
30.7692	AccountDetail.wsdl	http://localhost:8080/thesis/AccountDetail.wsdl

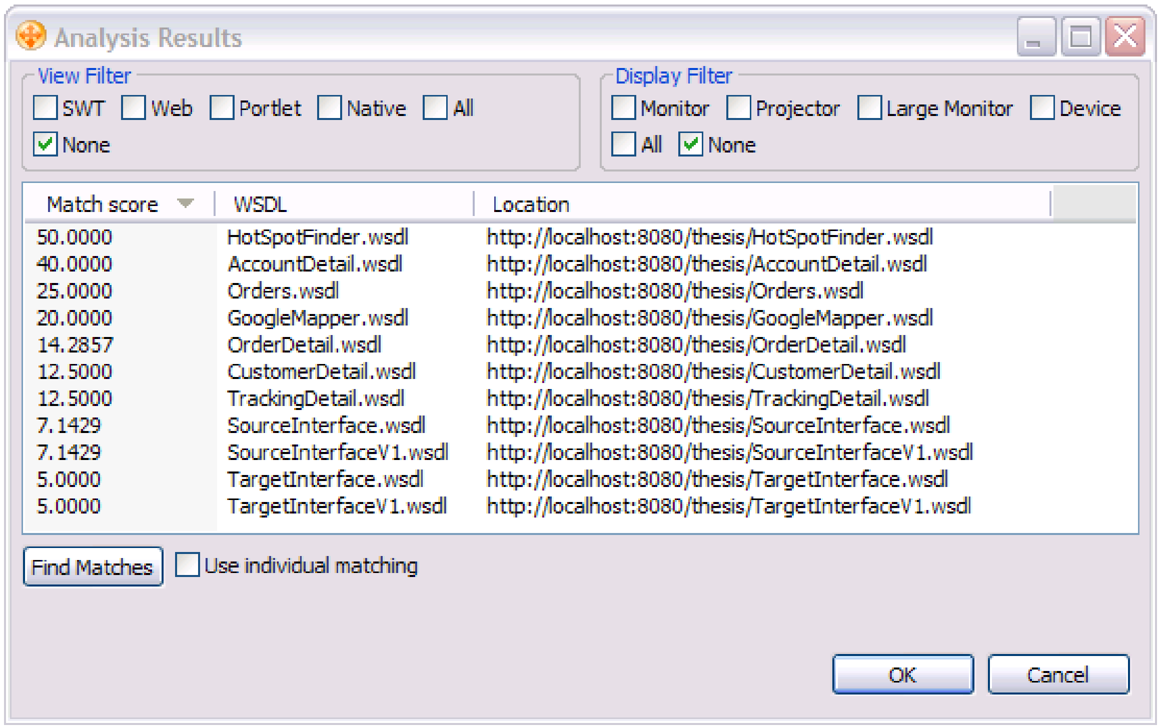
Use individual matching

Figure 21: Results of merged matching in experiment 2

5.4 Experiment Three – Assembling Travel Scenario via Matching

The third experiment involves building the complete travel RCP composite application. In this case, the target WSDLs will be the complete collection of WSDLs for both application scenarios and four additional WSDLs. The four additional WSDLs are provided as examples with the IBM Semantic Tools for Web Services [30]. These four WSDLs describe a collection of services related to the retail industry. Some of them contain semantic annotations and some of them do not. The full listing of these WSDLs can be found in Appendix D.

In order to start the RCP composite application, we must have a starting point. The CityState Picker component is added as the first component. The RCP composite application is then matched against the complete catalog of components. The results, as shown in Figure 22, tell us that the HotSpot Finder component has the highest score and should be added to the application.



View Filter: SWT [ ] Web [ ] Portlet [ ] Native [ ] All [ ] None [✓]

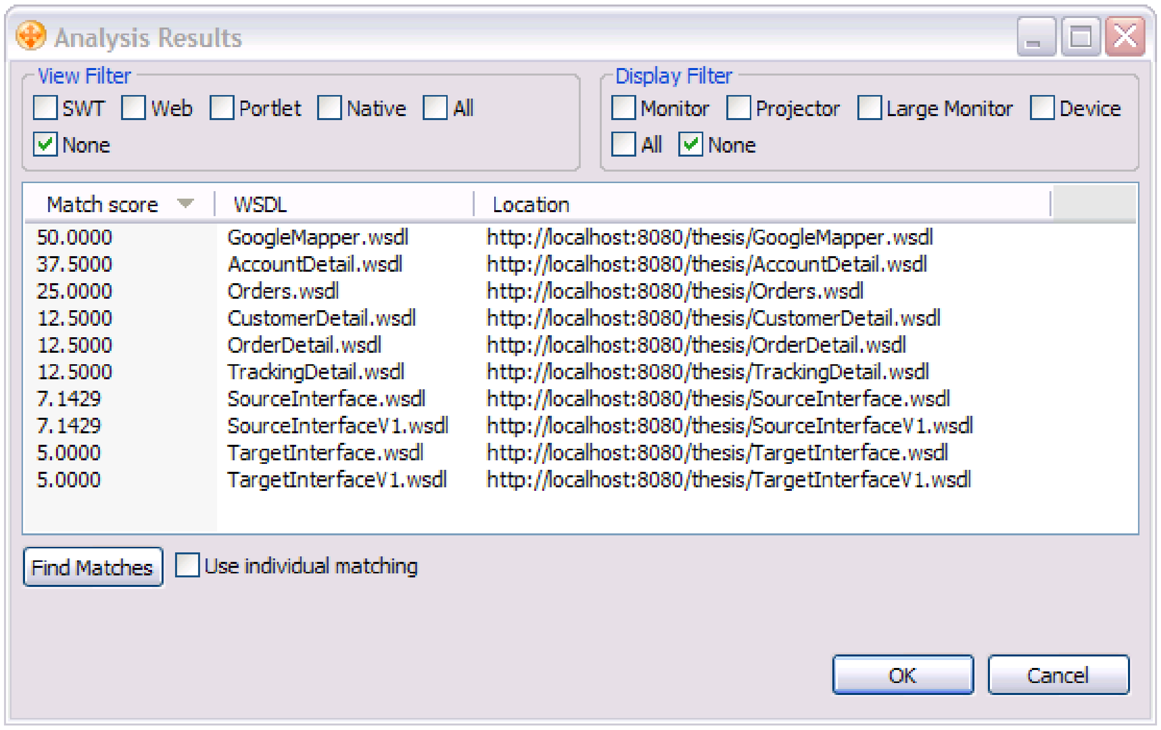
Display Filter: Monitor [ ] Projector [ ] Large Monitor [ ] Device [ ] All [ ] None [✓]

Match score	WSDL	Location
50.0000	HotSpotFinder.wsdl	http://localhost:8080/thesis/HotSpotFinder.wsdl
40.0000	AccountDetail.wsdl	http://localhost:8080/thesis/AccountDetail.wsdl
25.0000	Orders.wsdl	http://localhost:8080/thesis/Orders.wsdl
20.0000	GoogleMapper.wsdl	http://localhost:8080/thesis/GoogleMapper.wsdl
14.2857	OrderDetail.wsdl	http://localhost:8080/thesis/OrderDetail.wsdl
12.5000	CustomerDetail.wsdl	http://localhost:8080/thesis/CustomerDetail.wsdl
12.5000	TrackingDetail.wsdl	http://localhost:8080/thesis/TrackingDetail.wsdl
7.1429	SourceInterface.wsdl	http://localhost:8080/thesis/SourceInterface.wsdl
7.1429	SourceInterfaceV1.wsdl	http://localhost:8080/thesis/SourceInterfaceV1.wsdl
5.0000	TargetInterface.wsdl	http://localhost:8080/thesis/TargetInterface.wsdl
5.0000	TargetInterfaceV1.wsdl	http://localhost:8080/thesis/TargetInterfaceV1.wsdl

Use individual matching [ ]

Figure 22: Matching of CityPicker component

Once the HotSpot Finder component is added to the application, the analysis is run again. This time, as shown in Figure 23, the highest ranking component is the GoogleMapper component. In fact, not only is this component now the highest scoring component, but it has also shown a substantial jump from its previous score of 20 to the new score of 50. This result indicates that the GoogleMapper component is a good candidate to add to the application.


Analysis Results

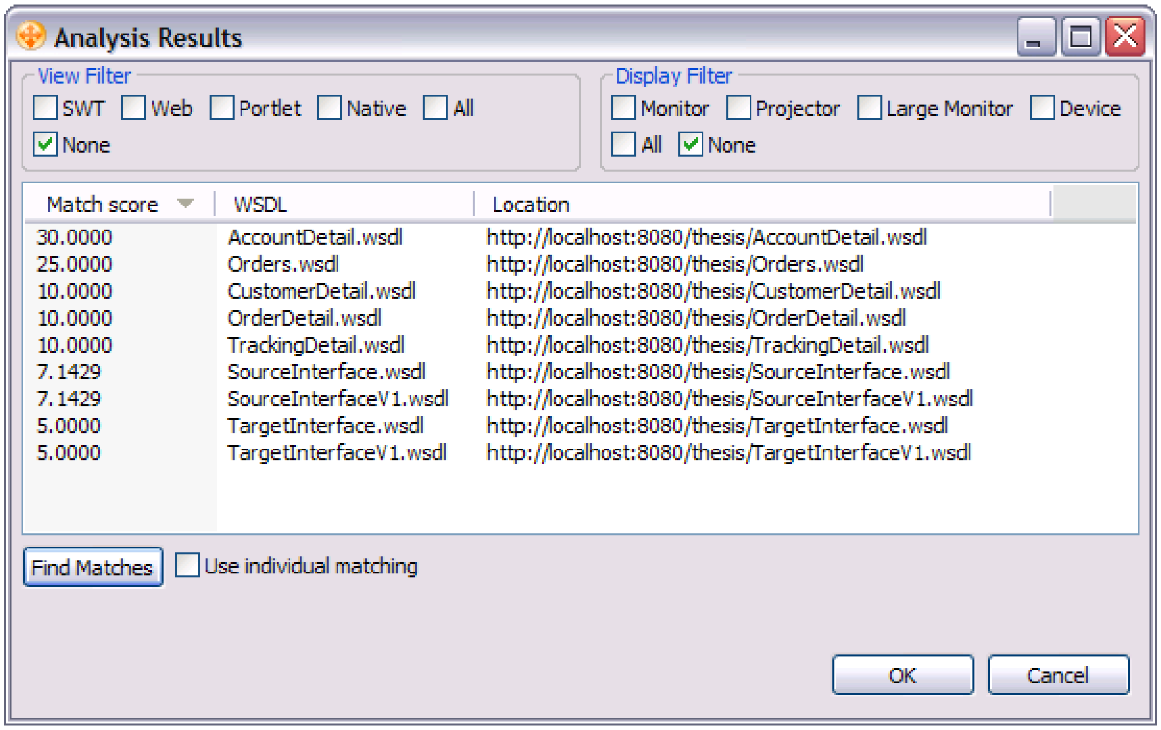
View Filter: [ ] SWT [ ] Web [ ] Portlet [ ] Native [ ] All [✓ None]

Display Filter: [ ] Monitor [ ] Projector [ ] Large Monitor [ ] Device [ ] All [✓ None]

Match score	WSDL	Location
50.0000	GoogleMapper.wsdl	http://localhost:8080/thesis/GoogleMapper.wsdl
37.5000	AccountDetail.wsdl	http://localhost:8080/thesis/AccountDetail.wsdl
25.0000	Orders.wsdl	http://localhost:8080/thesis/Orders.wsdl
12.5000	CustomerDetail.wsdl	http://localhost:8080/thesis/CustomerDetail.wsdl
12.5000	OrderDetail.wsdl	http://localhost:8080/thesis/OrderDetail.wsdl
12.5000	TrackingDetail.wsdl	http://localhost:8080/thesis/TrackingDetail.wsdl
7.1429	SourceInterface.wsdl	http://localhost:8080/thesis/SourceInterface.wsdl
7.1429	SourceInterfaceV1.wsdl	http://localhost:8080/thesis/SourceInterfaceV1.wsdl
5.0000	TargetInterface.wsdl	http://localhost:8080/thesis/TargetInterface.wsdl
5.0000	TargetInterfaceV1.wsdl	http://localhost:8080/thesis/TargetInterfaceV1.wsdl

Figure 23: Matching of CityPicker and HotSpot Finder

Once the GoogleMapper component is added to the application, the matching is run again. This time, as shown in Figure 24, the highest score is a 30 for the Account Details component. Looking back at the previous scores, we can see that the Account Details component score has been steadily decreasing. None of the other components has shown much increase in score. Therefore, we can conclude that there are no more appropriate components for this application. Using the matching process and incremental refinement, we have now been able to build the Travel applications, as show in Figure 16, section 4.2.



View Filter: SWT [ ] Web [ ] Portlet [ ] Native [ ] All [ ] [✓ None]

Display Filter: Monitor [ ] Projector [ ] Large Monitor [ ] Device [ ] All [ ] [✓ None]

Match score	WSDL	Location
30.0000	AccountDetail.wsdl	http://localhost:8080/thesis/AccountDetail.wsdl
25.0000	Orders.wsdl	http://localhost:8080/thesis/Orders.wsdl
10.0000	CustomerDetail.wsdl	http://localhost:8080/thesis/CustomerDetail.wsdl
10.0000	OrderDetail.wsdl	http://localhost:8080/thesis/OrderDetail.wsdl
10.0000	TrackingDetail.wsdl	http://localhost:8080/thesis/TrackingDetail.wsdl
7.1429	SourceInterface.wsdl	http://localhost:8080/thesis/SourceInterface.wsdl
7.1429	SourceInterfaceV1.wsdl	http://localhost:8080/thesis/SourceInterfaceV1.wsdl
5.0000	TargetInterface.wsdl	http://localhost:8080/thesis/TargetInterface.wsdl
5.0000	TargetInterfaceV1.wsdl	http://localhost:8080/thesis/TargetInterfaceV1.wsdl

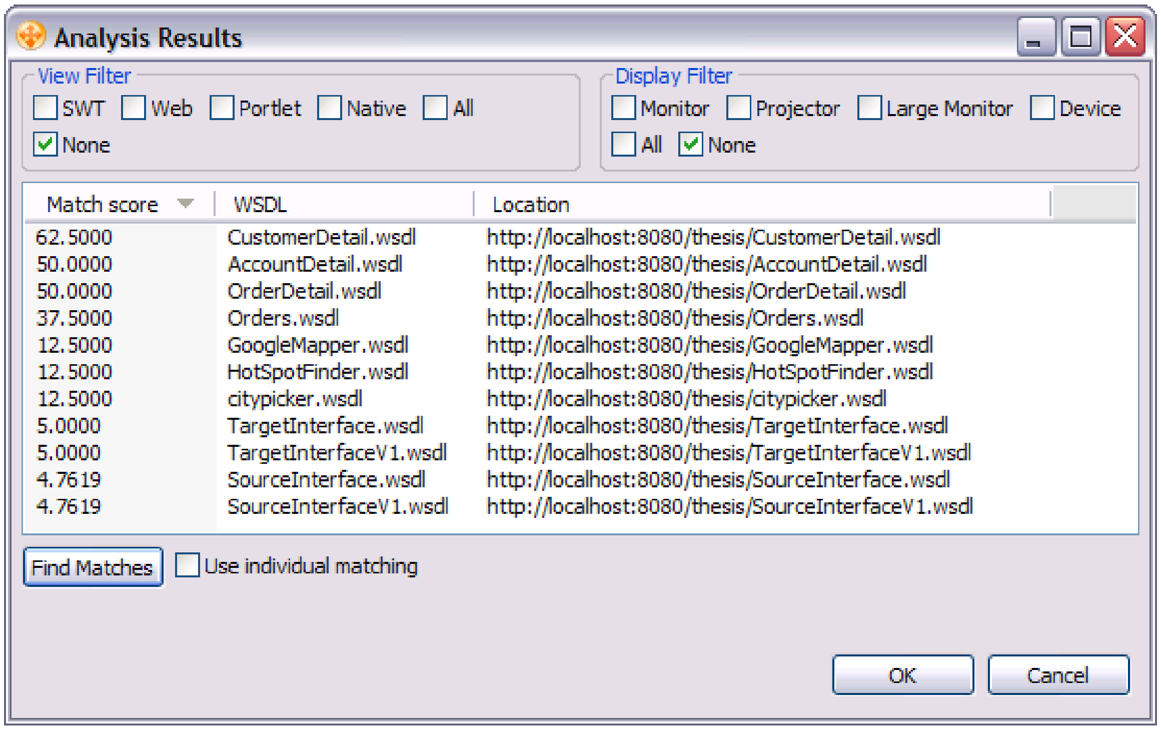
Use individual matching [ ]

Figure 24: Matching of CityState Picker, HotSpot Finder, and GoogleMapper

5.5 Experiment Four – Assembling the Order Tracking Scenario via Matching

The fourth experiment will show how the order tracking RCP composite application can be built using a similar process as that shown in experiment three. From looking at the possible starting points, the Orders component would be the most obvious one to use since it contains several outputs. However, instead we will use the Tracking Detail component to show how we can build the complete application. This is a reasonable choice to begin with because we are building an order "tracking" application.

The first step is to add the Tracking Detail component to the application and run the search. The results, as shown in Figure 25, tell us that the Customer Detail component would be a good one to add at this point.



View Filter: SWT [ ] Web [ ] Portlet [ ] Native [ ] All [ ] [✓ None]

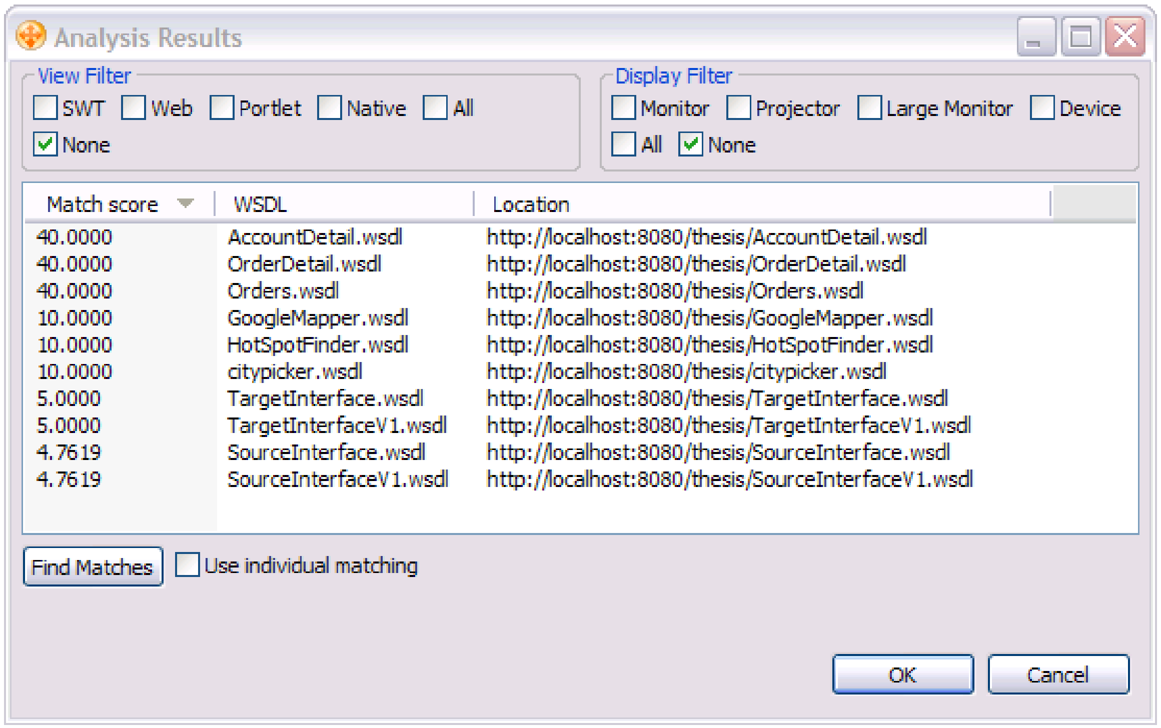
Display Filter: Monitor [ ] Projector [ ] Large Monitor [ ] Device [ ] All [ ] [✓ None]

Match score	WSDL	Location
62.5000	CustomerDetail.wsdl	http://localhost:8080/thesis/CustomerDetail.wsdl
50.0000	AccountDetail.wsdl	http://localhost:8080/thesis/AccountDetail.wsdl
50.0000	OrderDetail.wsdl	http://localhost:8080/thesis/OrderDetail.wsdl
37.5000	Orders.wsdl	http://localhost:8080/thesis/Orders.wsdl
12.5000	GoogleMapper.wsdl	http://localhost:8080/thesis/GoogleMapper.wsdl
12.5000	HotSpotFinder.wsdl	http://localhost:8080/thesis/HotSpotFinder.wsdl
12.5000	citypicker.wsdl	http://localhost:8080/thesis/citypicker.wsdl
5.0000	TargetInterface.wsdl	http://localhost:8080/thesis/TargetInterface.wsdl
5.0000	TargetInterfaceV1.wsdl	http://localhost:8080/thesis/TargetInterfaceV1.wsdl
4.7619	SourceInterface.wsdl	http://localhost:8080/thesis/SourceInterface.wsdl
4.7619	SourceInterfaceV1.wsdl	http://localhost:8080/thesis/SourceInterfaceV1.wsdl

Use individual matching [ ]

Figure 25: Matching with TrackingDetail

Once the Customer Detail component is added to the application, we can run the matching again. Figure 26 shows the results of this process. Looking at the scores, the average scores have decreased, though only by ten points. You will also observe that the score for the Orders component has actually increased by a small amount. Given that there are three possible components to choose from with equal scores, we will choose the Orders component because it has shown a consistent increase in value over the last two searches.



View Filter: SWT [ ] Web [ ] Portlet [ ] Native [ ] All [ ] None [✓]

Display Filter: Monitor [ ] Projector [ ] Large Monitor [ ] Device [ ] All [ ] None [✓]

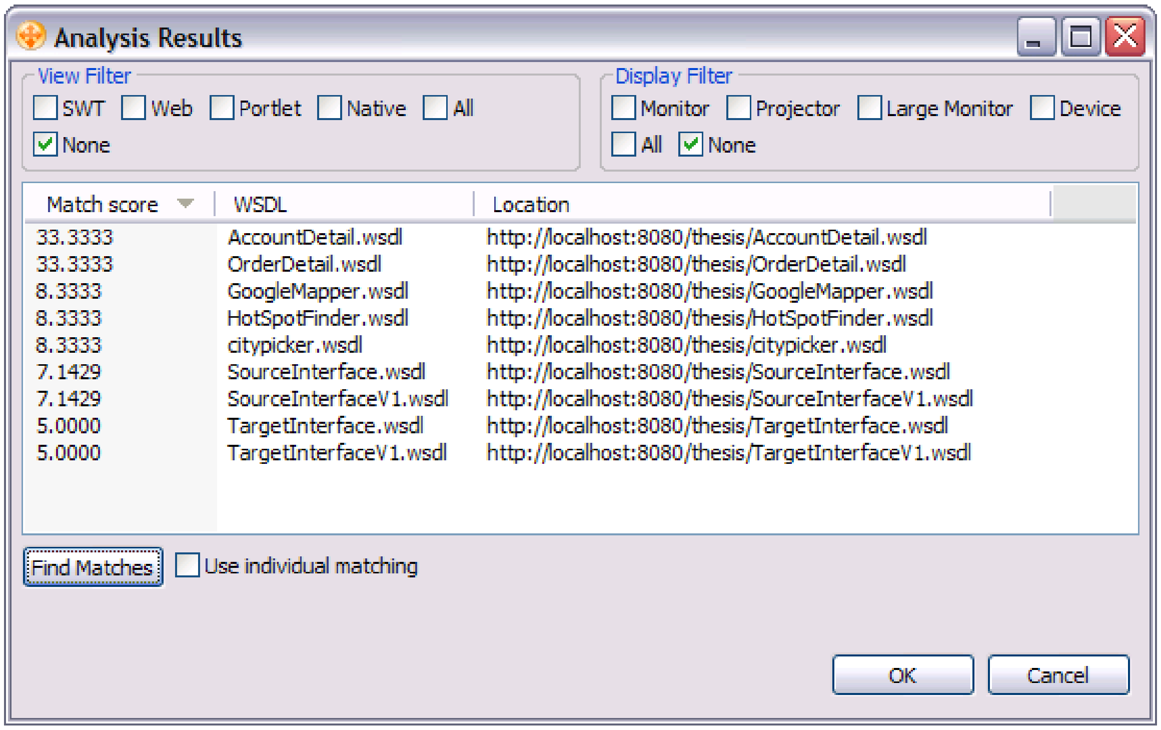
Match score	WSDL	Location
40.0000	AccountDetail.wsdl	http://localhost:8080/thesis/AccountDetail.wsdl
40.0000	OrderDetail.wsdl	http://localhost:8080/thesis/OrderDetail.wsdl
40.0000	Orders.wsdl	http://localhost:8080/thesis/Orders.wsdl
10.0000	GoogleMapper.wsdl	http://localhost:8080/thesis/GoogleMapper.wsdl
10.0000	HotSpotFinder.wsdl	http://localhost:8080/thesis/HotSpotFinder.wsdl
10.0000	citypicker.wsdl	http://localhost:8080/thesis/citypicker.wsdl
5.0000	TargetInterface.wsdl	http://localhost:8080/thesis/TargetInterface.wsdl
5.0000	TargetInterfaceV1.wsdl	http://localhost:8080/thesis/TargetInterfaceV1.wsdl
4.7619	SourceInterface.wsdl	http://localhost:8080/thesis/SourceInterface.wsdl
4.7619	SourceInterfaceV1.wsdl	http://localhost:8080/thesis/SourceInterfaceV1.wsdl

Use individual matching [ ]

Figure 26: Matching with TrackingDetail and CustomerDetail

The Orders component is added to the application and the matching analysis is run again. This time, in Figure 27, we observe that the overall scores have decreased again, but there is still a significant difference between the two highest scores and the third score. There is no real drive to choose one over the other based on the scores, so we need to choose one. Because we are building an order tracking application, the name Order Detail seems like a better choice than Account Detail. In other experiments not detailed here, it was seen that choosing the Account Detail component eventually lead to the same final RCP composite application described in this section. Additionally, the iterative nature of RCP composite application assembly allows assemblers to try out components and remove them if they do not prove to be useful. In this case, if the

Account Detail was found to not be usable in the application, the assembler could remove it and instead add the Order Detail component.



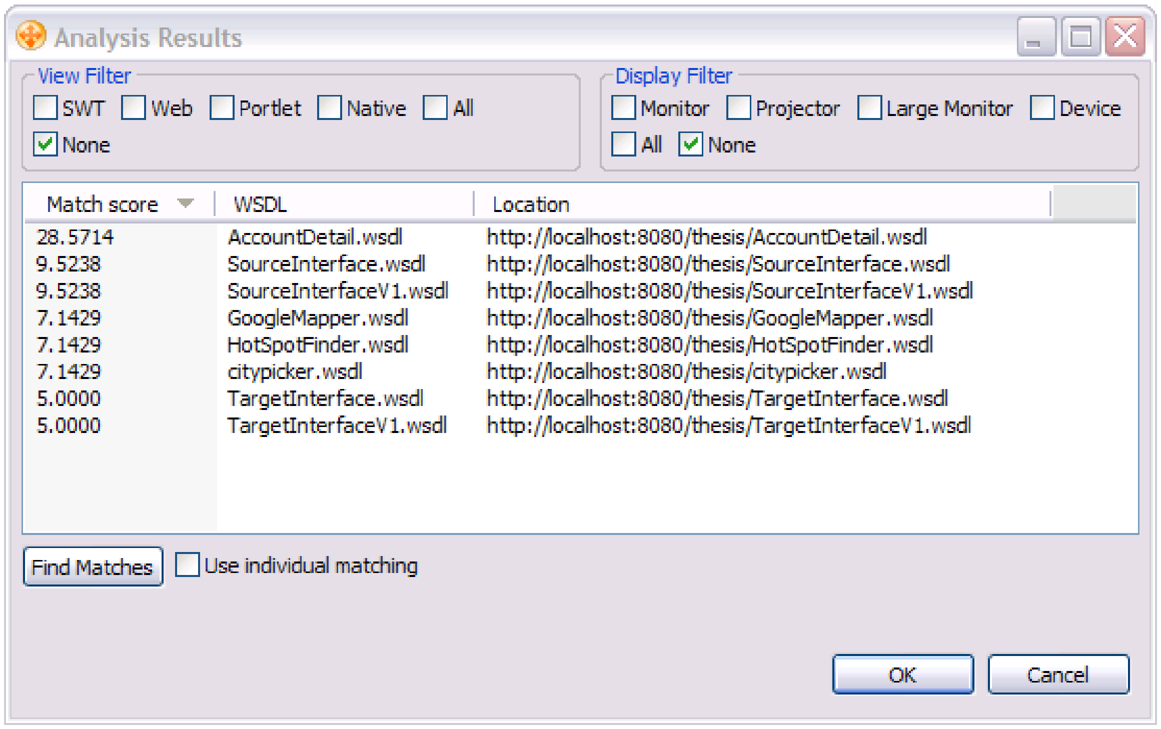
View Filter: [✓ None]

Display Filter: [✓ None]

Match score	WSDL	Location
33.3333	AccountDetail.wsdl	http://localhost:8080/thesis/AccountDetail.wsdl
33.3333	OrderDetail.wsdl	http://localhost:8080/thesis/OrderDetail.wsdl
8.3333	GoogleMapper.wsdl	http://localhost:8080/thesis/GoogleMapper.wsdl
8.3333	HotSpotFinder.wsdl	http://localhost:8080/thesis/HotSpotFinder.wsdl
8.3333	citypicker.wsdl	http://localhost:8080/thesis/citypicker.wsdl
7.1429	SourceInterface.wsdl	http://localhost:8080/thesis/SourceInterface.wsdl
7.1429	SourceInterfaceV1.wsdl	http://localhost:8080/thesis/SourceInterfaceV1.wsdl
5.0000	TargetInterface.wsdl	http://localhost:8080/thesis/TargetInterface.wsdl
5.0000	TargetInterfaceV1.wsdl	http://localhost:8080/thesis/TargetInterfaceV1.wsdl

Figure 27: Matching with TrackingDetail, CustomerDetail, and Orders

We add the Order Detail component to the application and run the analysis again. As we see in Figure 28, the top score has again dropped, but it is significantly higher than the other scores. We therefore decide to add the Account Detail component.



View Filter: SWT [ ] Web [ ] Portlet [ ] Native [ ] All [ ] [✓ None]

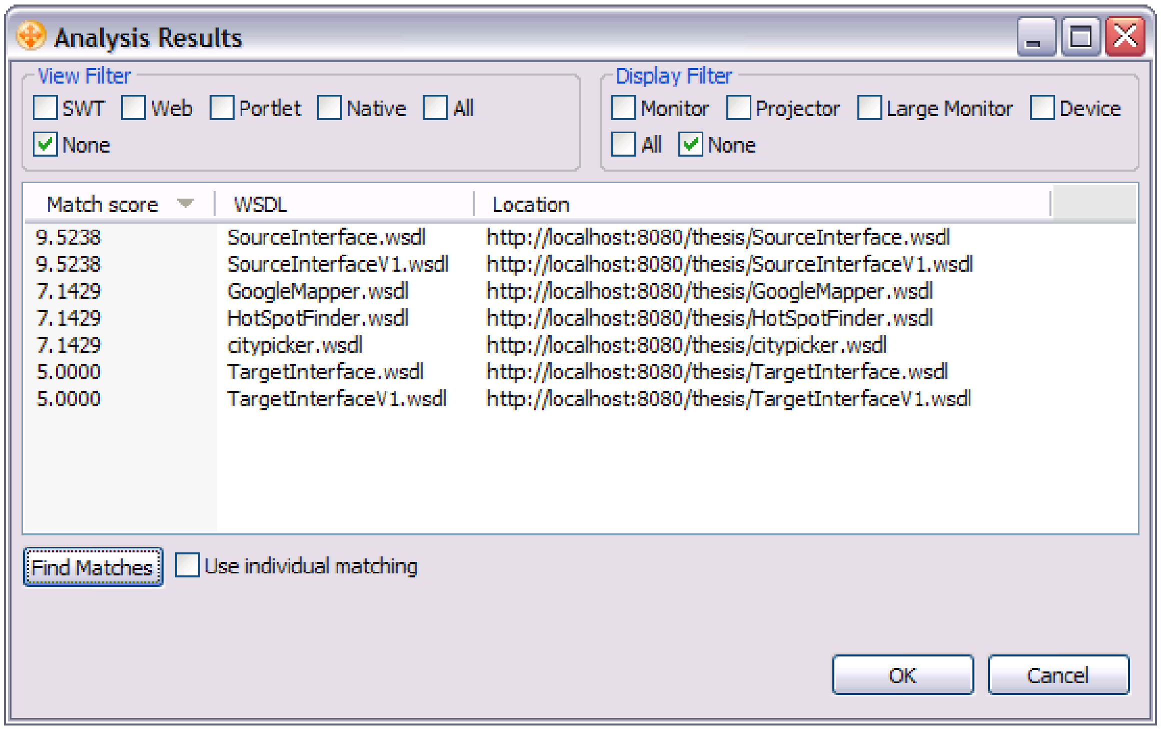
Display Filter: Monitor [ ] Projector [ ] Large Monitor [ ] Device [ ] All [ ] [✓ None]

Match score	WSDL	Location
28.5714	AccountDetail.wsdl	http://localhost:8080/thesis/AccountDetail.wsdl
9.5238	SourceInterface.wsdl	http://localhost:8080/thesis/SourceInterface.wsdl
9.5238	SourceInterfaceV1.wsdl	http://localhost:8080/thesis/SourceInterfaceV1.wsdl
7.1429	GoogleMapper.wsdl	http://localhost:8080/thesis/GoogleMapper.wsdl
7.1429	HotSpotFinder.wsdl	http://localhost:8080/thesis/HotSpotFinder.wsdl
7.1429	citypicker.wsdl	http://localhost:8080/thesis/citypicker.wsdl
5.0000	TargetInterface.wsdl	http://localhost:8080/thesis/TargetInterface.wsdl
5.0000	TargetInterfaceV1.wsdl	http://localhost:8080/thesis/TargetInterfaceV1.wsdl

Use individual matching [ ]

Figure 28: Matching with TrackingDetail, CustomerDetail, Orders, and OrderDetail

With the Account Detail component added to the application, we run the analysis again and get the results as shown in Figure 29. The top ranking score is now 9.5238 – much lower than what we started with and also much lower than the last component we added. It is reasonable to assume that the application is now complete. We can now customize the layout of the application to suit the user's needs based on the component we have selected. When completed, the GUI of the application can be seen in Figure 17 in section 4.3.



View Filter: [ ] SWT [ ] Web [ ] Portlet [ ] Native [ ] All [✓ None]

Display Filter: [ ] Monitor [ ] Projector [ ] Large Monitor [ ] Device [ ] All [✓ None]

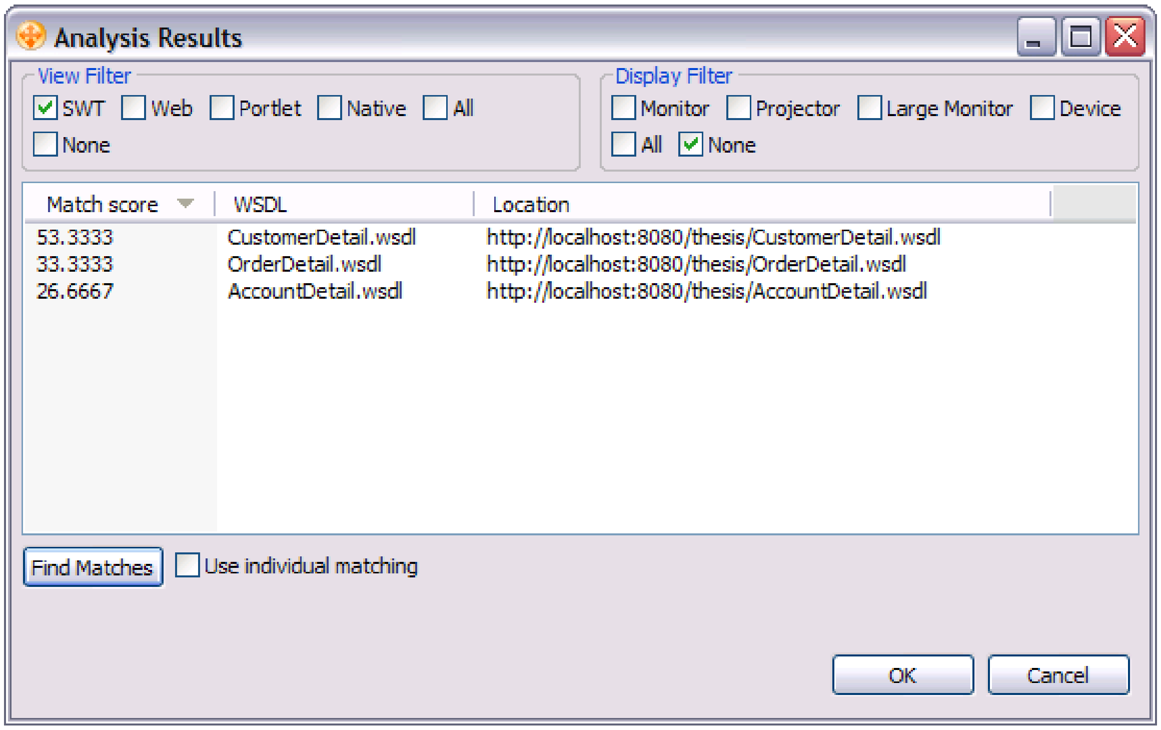
Match score	WSDL	Location
9.5238	SourceInterface.wsdl	http://localhost:8080/thesis/SourceInterface.wsdl
9.5238	SourceInterfaceV1.wsdl	http://localhost:8080/thesis/SourceInterfaceV1.wsdl
7.1429	GoogleMapper.wsdl	http://localhost:8080/thesis/GoogleMapper.wsdl
7.1429	HotSpotFinder.wsdl	http://localhost:8080/thesis/HotSpotFinder.wsdl
7.1429	citypicker.wsdl	http://localhost:8080/thesis/citypicker.wsdl
5.0000	TargetInterface.wsdl	http://localhost:8080/thesis/TargetInterface.wsdl
5.0000	TargetInterfaceV1.wsdl	http://localhost:8080/thesis/TargetInterfaceV1.wsdl

[ ] Use individual matching

Figure 29: Matching with All Order Detail components

5.6 Experiment Five – Adding GUI Characteristics to Matching

In this experiment we will show the effects of adding GUI information to the WSDL. In order to do this, a new message entry will be added to the search WSDL and each of the target WSDLs. For this experiment, the Customer Details component is marked as having an SWT UI, the Order Details component is marked as having a Portlet UI, and the Account Details is marked as having a Web UI. The test scenario is the same as experiment two with the merged WSDL search. Initially, the Orders and Order Tracking components are added to the application and a matching search is executed. By selecting one or more of the checkboxes at the top of the search dialog for SWT, Web, Portlet, Native, or All, the search WSDL will be enhanced with this additional criteria. In the first run we will select SWT checkbox. As can be seen in Figure 30, the match score for the Customer Detail component has increased slightly and the scores for the other components have decreased. Customer detail was originally the best match and it remains so with this additional filter. The original results can be found in Figure 21 in section 5.3.



View Filter: [✓ SWT] Web Portlet Native All None

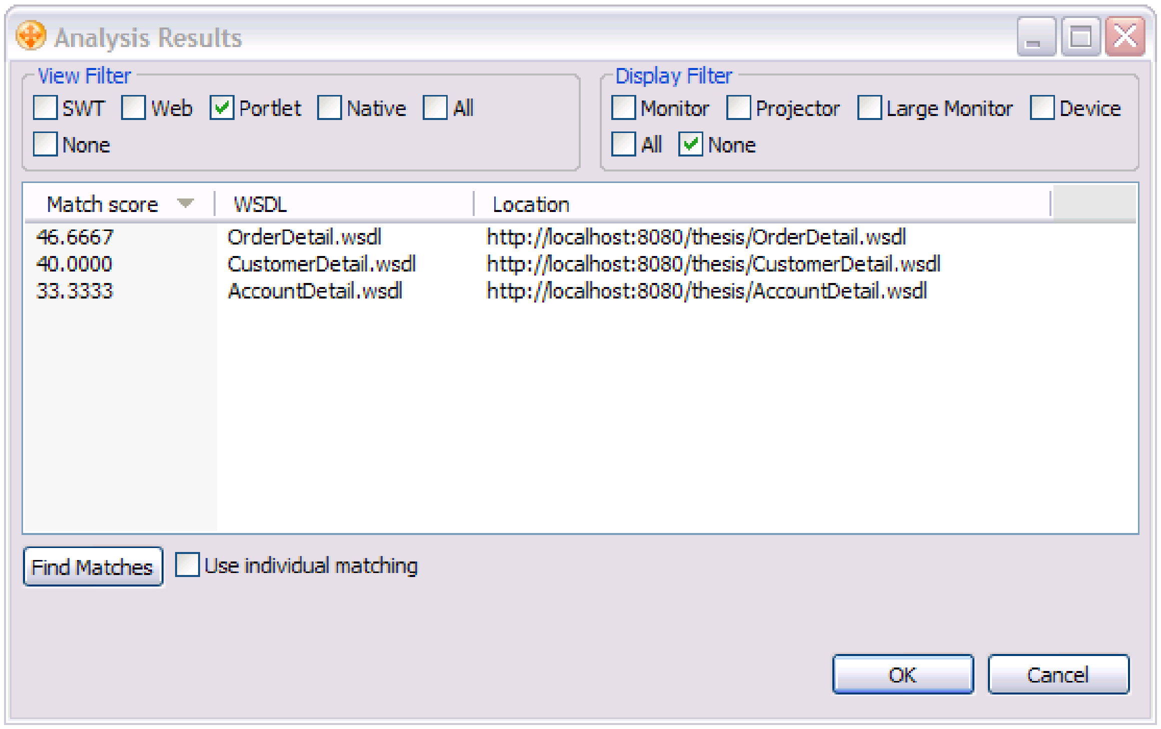
Display Filter: Monitor Projector Large Monitor Device All [✓ None]

Match score	WSDL	Location
53.3333	CustomerDetail.wsdl	http://localhost:8080/thesis/CustomerDetail.wsdl
33.3333	OrderDetail.wsdl	http://localhost:8080/thesis/OrderDetail.wsdl
26.6667	AccountDetail.wsdl	http://localhost:8080/thesis/AccountDetail.wsdl

Use individual matching

Figure 30: Search results with SWT selected as View Filter

In experiment two, the second highest score was returned by the Order Detail component. Now the matching search will be run again with Portlet selected as the View Filter, since Order Detail was marked in the repository as having this type of GUI. Not unexpectedly, this causes the score for the Order Detail component to rise, as seen in Figure 31, while the Customer Detail component shows a small drop in score. Interestingly, the Account Detail score rises slightly, from 30.7692 to 33.3333. This rise is discussed in more detail in the experiment analysis section below.



View Filter: [ ] SWT [ ] Web [✓ Portlet] [ ] Native [ ] All [ ] None

Display Filter: [ ] Monitor [ ] Projector [ ] Large Monitor [ ] Device [ ] All [✓ None]

Match score	WSDL	Location
46.6667	OrderDetail.wsdl	http://localhost:8080/thesis/OrderDetail.wsdl
40.0000	CustomerDetail.wsdl	http://localhost:8080/thesis/CustomerDetail.wsdl
33.3333	AccountDetail.wsdl	http://localhost:8080/thesis/AccountDetail.wsdl

[ ] Use individual matching

Figure 31: Search results with Portlet selected as View Filter

5.7 Experiment Analysis

Before any conclusions can be made regarding the experiments, the environment must be validated. Without validation, it is difficult to draw any meaningful conclusions. In order to validate the environment, experiment one was run using WSDLs that do not include the semantic annotations. Also, keywords such as city and state have been changed to random letter combinations. In this experiment, the matching score drops to 25. Additional changes to the WSDL that remove other keywords, but leave it otherwise functional cause the score to drop even more. This shows that the matching algorithm is working as expected in the experiments and that we have a valid environment.

Experiments one and two have shown that the function provided by existing web services matching code can be used in conjunction with RCP composite applications. Because the matching logic uses both text-based matching and semantic matching, the function can be used without adding the semantic markup. However, as we saw in experiment two, the semantic matching provides better results. For example, when two components are named differently yet provide the same functionality, the semantic matching is able to find the match.

Experiments three and four have shown that it is possible to build a RCP composite application from a collection of different components using semantic annotations and semantic web service matching logic. While there is no automatic way to start the building process, once a starting point is selected, the remaining compatible components begin to stand out in the repository searches. As the travel scenario application was composed, the initial results did not show as much difference in scores as might be expected, but with the addition of the HotSpot Finder component, the GoogleMapper component jumped out as the next obvious addition. With the assembly of the Order Tracking application, the components that could be added to this application really stood out with scores three or more times greater for the expected components. In both of these cases, the additional Source and Target WSDLs that were added to the repository continue to score low in all cases. This is not surprising as these components describe functions unrelated to the applications being built.

Experiment five provides interesting results that require additional analysis to understand. What is immediately clear is that the addition of the View Filter to the search input and the candidate components does improve the score for those components with matching view types. What is not as clear is why one component, Account Detail, increased in score when it did not match the given filter. Through closer examination of the algorithm, the score increase can be accounted for by the fact that the Account Detail component specified a view type. The addition of the view element in the search WSDL and all UI components in the repository caused the matching algorithm to detect an additional potential match. Therefore, the score increases slightly because the view is specified, but it does not rise as much as the component that had a matching GUI view type of Portlet. The experiments have shown that the score of a component can be impacted by the addition of the view element. It can therefore be surmised that any potential metadata could be added to the WSDL and used in the semantic matching logic.

So, does the matching code work as expected with the additional view element? The answer is, not surprisingly: It depends on what is expected. The addition of the new metadata is treated at the same level of importance as the other components of the WSDL. This may or may not be what is expected. The selection of a view type as a filter could be meant as a preferred view. In this case, because of the equal weighting, the view types filter works as a preferred filter. If the assembler is stating their preference for a particular interface type, then the filter function is working as expected. However, this selection of a filter could also be a hard requirement of the RCP composite application. This means that if selected, only these types of components should be available to the application. If this is the expected result, then the filtering logic would need to be applied differently. Rather than including the View Filter in the search WSDL, the target WSDLs could be processed prior to the matching logic. Before the resource-intensive matching logic is run, the list of target WSDLs could be filtered to not include any components that do not meet the restriction. In this case, the element would provide an all-or-nothing match, meaning only components that specify the selected view type would be run through the matching logic and be available as possible matches. Since the view is represented in the WSDL as an element with semantic annotations, finding the appropriate elements to discard would require only a single pass through each of the potential target WSDL files.

Chapter VI

CONCLUSIONS

6.1 Findings

The assembly of RCP composite applications from a catalog of components can be a difficult task to accomplish, particularly for non-programmers who would like to assemble RCP composite applications. One of the most difficult problems faced by these users is finding compatible and complementary components in a large catalog of components that have been built by different groups, at different times, using different technologies and programming conventions. This thesis demonstrated that this problem can be largely solved by applying technologies related to the semantic web and web services matching. Because the programmatic inputs and outputs of components are already described using WSDL, it is a natural extension to apply web services technologies to these components.

The first technology that can be applied is Semantic Annotations for WSDL (SAWSDL), as standardized by the W3C. By adding semantic model references to the message elements of the WSDL, the properties exposed by the component can be better described using modeling languages. Since the modeling attributes can be added to the other elements of the WSDL, the definition of the component could be further refined and described using the concepts of SAWSDL.

The second technology group that can be applied is the searching and matching algorithms created for use with web services. These algorithms provide a powerful method for scoring the compatibility of an RCP component from a large set of possible component choices. This scoring simplifies the application creation process for the RCP composite application assembler by providing a ranking of potential components. This allows the assembler to focus on the highest ranked components, skipping over the lower ranked components, when considering which items may be compatible in the application being created.

The searching process is further improved based on the fact that a RCP composite application can be viewed and described as a single component when searching against a repository of components. This is done by creating a merged WSDL from each of the member WSDLs of the RCP composite application. In order to produce the most valuable results, the merged WSDL is filtered to not include duplicate elements. As seen in the experiment results, the use of individual matching may still be valuable, especially when attempting to distinguish between components that score very closely to each other. A potential improvement to the analysis results window, as shown in this thesis, would be to display the score for each target component using both the merged matching and individual matching, when the collection of scores is relatively close.

Finally, it has been shown that additional characteristics of components can be described using the WSDL file and that additional semantic annotations of those properties can be used to enhance the search results. When GUI descriptions are added to the WSDL with the appropriate semantic annotations, the matching logic was able to consider these additional attributes when searching for compatible components. Through a slight modification of the search logic, it may be possible to change this filtering selection from an equally weighted attribute to an all-or-nothing attribute. In the all-or-nothing approach, if the selected view is not described in the target component, it would not be considered for the application.

6.2 Feasibility

An important aspect to consider when using the methods described in this thesis is how feasible this solution is. In general, the process of creating components is not an easy one. It requires creating chunks of software that can potentially be wired to all sorts of other pieces of software. Since the component developer has no prior knowledge of these other pieces of potential software, creating and describing meaningful inputs and outputs is difficult. The choice of names may seem like a minor issue when creating the component, but this becomes a very important piece of information to a RCP composite application assembler. That said, the additional work necessary to annotate the components for matching as described in this thesis is only a minimal amount of delta work.

The choice of using RDF or OWL as the modeling language makes sense when looking at the entire semantic web. When focusing directly on RCP composite applications, this may not make the most sense. A better choice of modeling languages may be the Unified Modeling Language (UML) [31]. Although RDF and OWL provide powerful features for describing relationships, these languages and features are not generally used or necessary when creating components for RCP composite applications. UML diagrams, on the other hand, are commonly used to architect software components designed for SOA and reuse. One could argue that RDF and OWL are better choices because they are designed to allow reasoning about relationships between objects in the models. However, it has been shown by A. Evans [32] that reasoning can be done with UML class diagrams. Additionally, UML is easily generated from existing code. Therefore, it is not necessary to create the UML separately if it is created as part of the architecture. Further, the UML could be automatically generated using reverse analysis techniques and a bottom-up approach. While these options may also be possible with RDF and OWL, it is not nearly as common or widely used. Alternatively, it has also been shown [33] that OWL models can be generated from UML. This may provide another practical and automated approach to using ontologies with components for RCP composite applications.

Another potential method of automating the process would be to automatically inject the view type elements into the WSDL. In order to build a component, the developer must associate the properties and actions in the WSDL file with a Java class. Using this mapping and the support for introspection provided by the Java language, the class can be analyzed to determine the type of view that it provides. If the view extends the Eclipse ViewPart class, it is reasonable to assume that the view type is SWT. If the Java class extends the Browser, AWT classes, or Swing classes, it would be reasonable to assume the view type and inject the appropriate data into the WSDL. This could be done at development time, or a batch process could be run across the complete catalog to analyze all member components and be updated as needed.

6.3 Composite Application Usefulness

Composite Applications look to become more useful and popular as the ability to reuse existing components is improved. By allowing non-technical assemblers to create applications on the fly, as-needed, business has the ability to leverage existing assets in new ways. Simple changes like automatically duplicating entries between application components reduces the amount of time it takes to enter data and reduces the possibilities of introducing data errors into the system. This time and error savings should provide to save a significant amount of expense savings over existing applications.

As tooling improves, the ability to create new components will be simplified. By simplifying the development process, more reusable components can be constructed to grow the number of available components. This should allow businesses to create a tool chest of reusable assets that can be deployed and assembled by non-technical users, when needed, to solve immediate business needs. This can further lead to a reduction in the time needed to solve customer problems, complete projects, and reduce custom development expense.

Assembling applications remains one of the key drivers to the success of RCP composite applications. Non-technical users can quickly build and wire applications from existing components. If a component does not fit into a specific application or scenario, it can just as easily be removed from the application. The iterative approach to building RCP composite applications allows assemblers to replace components when different, perhaps more advanced, components become available.

6.4 Future Direction

As with web services and EJBs, the number, diversity, and complexity of components will continue to expand. As more businesses begin to see the value provided by allowing non-technical employees to compose applications to solve business needs, the number and types of components will grow. In order to handle this growth, new repositories, cataloging techniques, and validation processes will be needed. For the non-technical person to understand these components, they will need to be self explanatory.

The work being done in the web services world can be used as a foundation for RCP composite application components. Catalogs similar to UDDI registries will be needed to allow developers and assemblers to store, catalog, and find components. However, given that non-technical users will need to access these catalogs, the interface used for matching will need to be simplified.

As has been shown, modeling of components using ontologies and other semantic web technologies will make it easier for non-technical users to find components. In order to make this matching even more valuable to end users, standard industry ontologies will need to be developed and applied to components (for example, GUI ontologies). Additionally, the current algorithms may need to be improved. The improvements may include: 1) allowing the user to specify only a specific set of inputs or outputs to consider; 2) allowing a user to specify a weight on certain sets of inputs or outputs that affect the overall score of the matching; 3) providing ways to better generate the ontologies based on the component code and a user's knowledge of a particular industry; and 4) using only inputs and outputs that are not currently being utilized in the RCP composite application.

APPENDIX A

```
<?xml version="1.0" encoding="UTF-8"?>
<definitions name="edu.txstate.mpc"
  targetNamespace="http://www.ibm.com/wps/c2a/edu.txstate.mpc"
  xmlns="http://schemas.xmlsoap.org/wsdl/"
  xmlns:TravelOnt="http://localhost:8080/thesis/travel.owl"
  xmlns:portlet="http://www.ibm.com/wps/c2a"
  xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
  xmlns:tns="http://www.ibm.com/wps/c2a/edu.txstate.mpc"
  xmlns:wssem="http://www.w3.org/ns/sawsdl"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <types>
    <xsd:schema targetNamespace="http://www.ibm.com/wps/c2a/edu.txstate.mpc"/>
  </types>
  <message name="cityState">
    <part name="cityState" type="xsd:string" wssem:modelReference="TravelOnt#City"/>
  </message>
  <portType name="edu.txstate.mpc_Service">
    <operation name="pubCityState">
      <output message="tns:cityState"/>
    </operation>
  </portType>
  <binding name="edu.txstate.mpcbinding" type="tns:edu.txstate.mpc_Service">
    <portlet:binding/>
    <operation name="pubCityState">
      <portlet:action activeOnStartup="true" caption="pubCityState"
        description="Announces the city and state" name="pubCityState"
        selectOnMultipleMatch="false" type="standard"/>
      <output>
        <portlet:param boundTo="request-attribute" caption="cityState"
          description="Published ths city/state selected"
          name="cityState" partname="cityState"/>
      </output>
    </operation>
  </binding>
</definitions>
```

CityStatePicker.wsdl

```xml
<?xml version="1.0" encoding="UTF-8"?>
<definitions name="edu.txstate.mpc"
  targetNamespace="http://www.ibm.com/wps/c2a/edu.txstate.mpc"
  xmlns="http://schemas.xmlsoap.org/wsdl/"
  xmlns:TravelOnt="http://localhost:8080/thesis/travel.owl"
  xmlns:portlet="http://www.ibm.com/wps/c2a"
  xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
  xmlns:tns="http://www.ibm.com/wps/c2a/edu.txstate.mpc"
  xmlns:wssem="http://www.w3.org/ns/sawsdl"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <types>
    <xsd:schema targetNamespace="http://www.ibm.com/wps/c2a/edu.txstate.mpc"/>
  </types>
  <message name="City">
    <part name="City" type="xsd:string" wssem:modelReference="TravelOnt#City" />
  </message>
  <message name="Address">
    <part name="Address" type="xsd:string" wssem:modelReference="TravelOnt#Address" />
  </message>
  <portType name="edu.txstate.mpc_Service">
    <operation name="SetLocationCityState">
      <input message="tns:City"/>
    </operation>
    <operation name="PubAddress">
      <output message="tns:Address"/>
    </operation>
  </portType>
  <binding name="edu.txstate.mpcbinding" type="tns:edu.txstate.mpc_Service">
    <portlet:binding/>
    <operation name="SetLocationCityState">
      <portlet:action activeOnStartup="true"
        caption="SetLocationCityState" description=""
        name="SetLocationCityState" selectOnMultipleMatch="false" type="standard"/>
      <input>
        <portlet:param boundTo="request-attribute" name="City" partname="City"/>
      </input>
    </operation>
    <operation name="PubAddress">
      <portlet:action activeOnStartup="true" caption="" description=""
        name="PubAddress" selectOnMultipleMatch="false" type="standard"/>
      <output>
        <portlet:param boundTo="request-attribute" name="Address" partname="Address"/>
      </output>
    </operation>
  </binding>
</definitions>
```

HotSpotFinder.wsdl

```
<?xml version="1.0" encoding="UTF-8"?>
<definitions name="edu.txstate.mpc.GoogleMapper"
  targetNamespace="http://www.ibm.com/wps/c2a/edu.txstate.mpc"
  xmlns="http://schemas.xmlsoap.org/wsdl/"
  xmlns:portlet="http://www.ibm.com/wps/c2a"
  xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
  xmlns:tns="http://www.ibm.com/wps/c2a/edu.txstate.mpc"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:wssem="http://www.w3.org/ns/sawsdl"
  xmlns:TravelOnt="http://localhost:8080/thesis/travel.owl">
  <types>
    <xsd:schema targetNamespace="http://www.ibm.com/wps/c2a/edu.txstate.mpc"/>
  </types>
  <message name="Address">
    <part name="Address" type="xsd:string"
        wssem:modelReference="TravelOnt#Address"/>
  </message>

  <portType name="edu.txstate.mpc.Address_Service">
    <operation name="SetAddress">
      <input message="tns:Address"/>
    </operation>
  </portType>

  <binding name="edu.txstate.mpc.Addressbinding" type="tns:edu.txstate.mpc.Address_Service">
    <portlet:binding/>
    <operation name="SetAddress">
      <portlet:action activeOnStartup="true" caption="SetAddress"
        description="Sets the address to be shown in the view"
        name="SetAddress" selectOnMultipleMatch="false" type="standard"/>
      <input>
        <portlet:param boundTo="request-attribute" caption="Address"
          description="Sets the address to be displayed"
          name="Address" partname="Address"/>
      </input>
    </operation>
  </binding>
</definitions>
```
GoogleMapper.wsdl

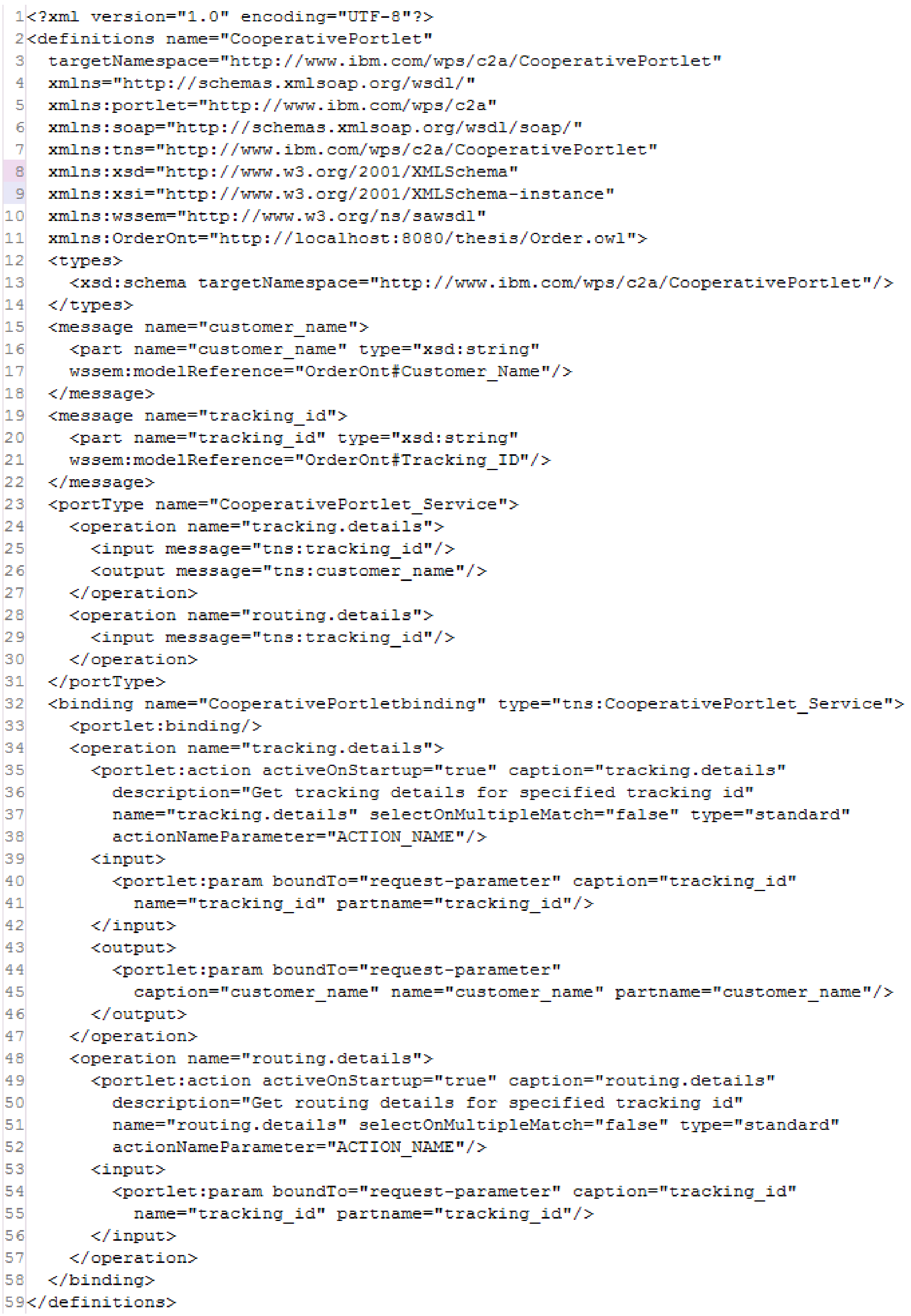
APPENDIX B

```xml
<?xml version="1.0" encoding="UTF-8"?>
<definitions name="CooperativePortlet"
    targetNamespace="http://www.ibm.com/wps/c2a/CooperativePortlet"
    xmlns="http://schemas.xmlsoap.org/wsdl/"
    xmlns:portlet="http://www.ibm.com/wps/c2a"
    xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
    xmlns:tns="http://www.ibm.com/wps/c2a/CooperativePortlet"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:wssem="http://www.w3.org/ns/sawsdl"
    xmlns:OrderOnt="http://localhost:8080/thesis/Order.owl">
    <types>
        <xsd:schema
            targetNamespace="http://www.ibm.com/wps/c2a/CooperativePortlet" />
    </types>
    <message name="order_month">
        <part name="order_month" type="xsd:string"
        wssem:modelReference="OrderOnt#Order_Month"/>
    </message>
    <message name="OrderMonthResponse">
        <part name="order_id" type="xsd:string"
        wssem:modelReference="OrderOnt#Order_ID" />
        <part name="customer_id" type="xsd:string"
        wssem:modelReference="OrderOnt#Customer_ID" />
    </message>
    <portType name="CooperativePortlet_Service">
        <operation name="SHIPPINGORDERSordersForMonth">
            <input message="tns:order_month" />
            <output message="tns:OrderMonthResponse" />
        </operation>
    </portType>
    <binding name="CooperativePortletbinding"
        type="tns:CooperativePortlet_Service">
        <portlet:binding />
        <operation name="SHIPPINGORDERSordersForMonth">
            <portlet:action activeOnStartup="true"
                caption="orders.for.month"
                description="get orders for specified month"
                name="SHIPPINGORDERSordersForMonth" selectOnMultipleMatch="false"
                type="standard" actionNameParameter="ACTION_NAME">
                <portlet:constant-params>
                    <portlet:constant-param name="defaultMonth"
                        value="January" />
                </portlet:constant-params>
            </portlet:action>
            <input>
                <portlet:param boundTo="request-parameter"
                    caption="order_month" name="order_month" partname="order_month" />
            </input>
            <output>
                <portlet:param boundTo="request-parameter"
                    caption="order_id" name="order_id" partname="order_id" />
                <portlet:param boundTo="request-parameter"
                    name="customer_id" partname="customer_id" />
            </output>
        </operation>
    </binding>
</definitions>
```

Orders.wsdl

```
<?xml version="1.0" encoding="UTF-8"?>
<definitions name="CooperativePortlet"
  targetNamespace="http://www.ibm.com/wps/c2a/CooperativePortlet"
  xmlns="http://schemas.xmlsoap.org/wsdl/"
  xmlns:portlet="http://www.ibm.com/wps/c2a"
  xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
  xmlns:tns="http://www.ibm.com/wps/c2a/CooperativePortlet"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:wssem="http://www.w3.org/ns/sawsdl"
  xmlns:OrderOnt="http://localhost:8080/thesis/Order.owl">
  <types>
    <xsd:schema targetNamespace="http://www.ibm.com/wps/c2a/CooperativePortlet"/>
  </types>
  <message name="order_id">
    <part name="order_id" type="xsd:string"
    wssem:modelReference="OrderOnt#Order_ID"/>
  </message>
  <message name="tracking_id">
    <part name="tracking_id" type="xsd:string"
    wssem:modelReference="OrderOnt#Tracking_ID"/>
  </message>
  <portType name="CooperativePortlet_Service">
    <operation name="order.details">
      <input message="tns:order_id"/>
      <output message="tns:tracking_id"/>
    </operation>
  </portType>
  <binding name="CooperativePortletbinding" type="tns:CooperativePortlet_Service">
    <portlet:binding/>
    <operation name="order.details">
      <portlet:action activeOnStartup="true" caption="order.details"
        description="Get details for specified order id"
        name="order.details" selectOnMultipleMatch="false" type="standard"
        actionNameParameter="ACTION_NAME"/>
      <input>
        <portlet:param boundTo="request-parameter" caption="order_id"
          name="order_id" partname="order_id"/>
      </input>
      <output>
        <portlet:param boundTo="request-parameter" caption="tracking_id"
          name="tracking_id" partname="tracking_id"/>
      </output>
    </operation>
  </binding>
</definitions>
```

OrderDetail.wsdl

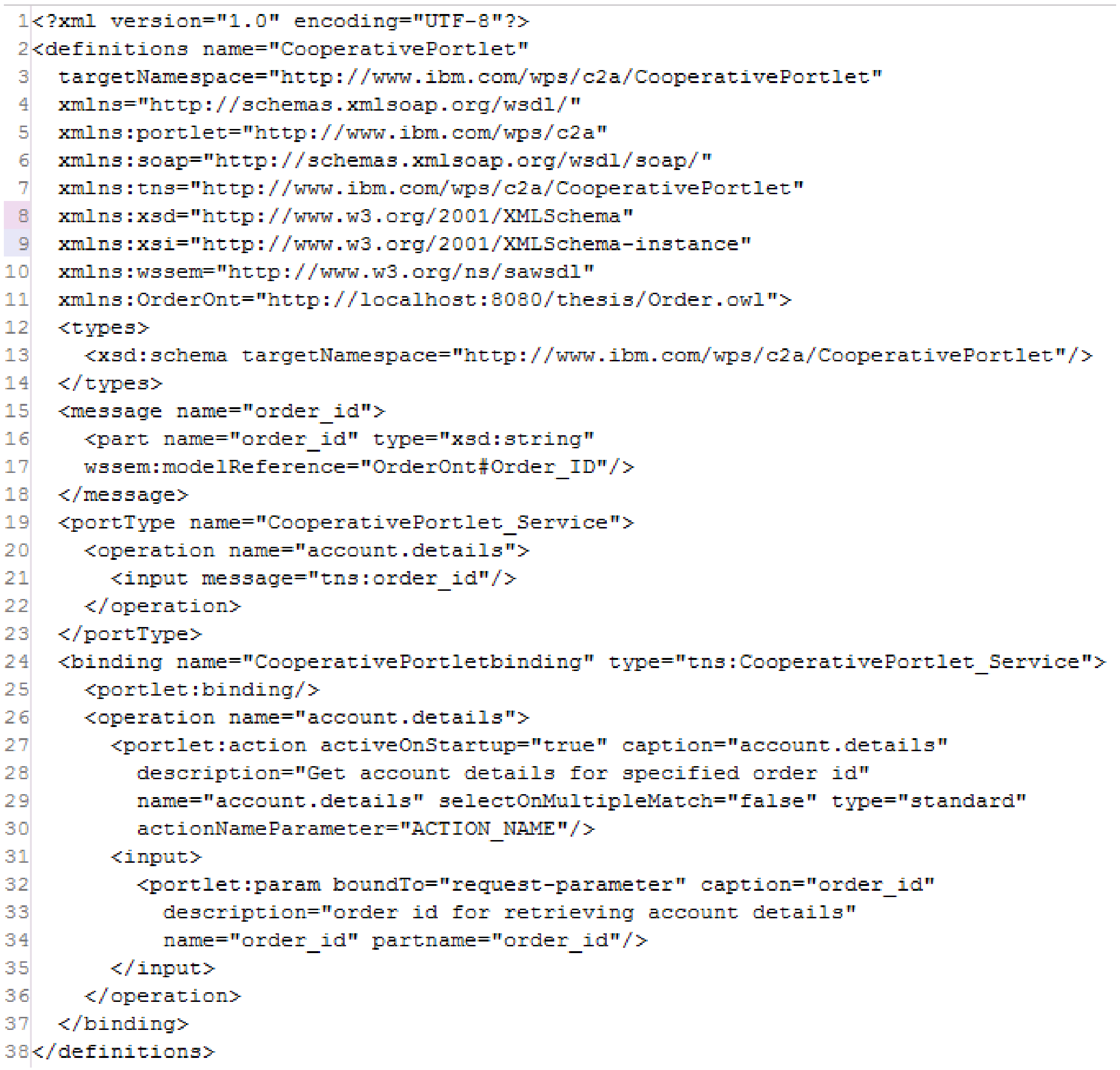


```xml
<?xml version="1.0" encoding="UTF-8"?>
<definitions name="CooperativePortlet"
  targetNamespace="http://www.ibm.com/wps/c2a/CooperativePortlet"
  xmlns="http://schemas.xmlsoap.org/wsdl/"
  xmlns:portlet="http://www.ibm.com/wps/c2a"
  xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
  xmlns:tns="http://www.ibm.com/wps/c2a/CooperativePortlet"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:wssem="http://www.w3.org/ns/sawsdl"
  xmlns:OrderOnt="http://localhost:8080/thesis/Order.owl">
  <types>
    <xsd:schema targetNamespace="http://www.ibm.com/wps/c2a/CooperativePortlet"/>
  </types>
  <message name="customer_name">
    <part name="customer_name" type="xsd:string"
    wssem:modelReference="OrderOnt#Customer_Name"/>
  </message>
  <message name="tracking_id">
    <part name="tracking_id" type="xsd:string"
    wssem:modelReference="OrderOnt#Tracking_ID"/>
  </message>
  <portType name="CooperativePortlet_Service">
    <operation name="tracking.details">
      <input message="tns:tracking_id"/>
      <output message="tns:customer_name"/>
    </operation>
    <operation name="routing.details">
      <input message="tns:tracking_id"/>
    </operation>
  </portType>
  <binding name="CooperativePortletbinding" type="tns:CooperativePortlet_Service">
    <portlet:binding/>
    <operation name="tracking.details">
      <portlet:action activeOnStartup="true" caption="tracking.details"
        description="Get tracking details for specified tracking id"
        name="tracking.details" selectOnMultipleMatch="false" type="standard"
        actionNameParameter="ACTION_NAME"/>
      <input>
        <portlet:param boundTo="request-parameter" caption="tracking_id"
          name="tracking_id" partname="tracking_id"/>
      </input>
      <output>
        <portlet:param boundTo="request-parameter"
          caption="customer_name" name="customer_name" partname="customer_name"/>
      </output>
    </operation>
    <operation name="routing.details">
      <portlet:action activeOnStartup="true" caption="routing.details"
        description="Get routing details for specified tracking id"
        name="routing.details" selectOnMultipleMatch="false" type="standard"
        actionNameParameter="ACTION_NAME"/>
      <input>
        <portlet:param boundTo="request-parameter" caption="tracking_id"
          name="tracking_id" partname="tracking_id"/>
      </input>
    </operation>
  </binding>
</definitions>
```

TrackingDetail.wsdl

```xml
<?xml version="1.0" encoding="UTF-8"?>
<definitions name="CooperativePortlet"
  targetNamespace="http://www.ibm.com/wps/c2a/CooperativePortlet"
  xmlns="http://schemas.xmlsoap.org/wsdl/"
  xmlns:portlet="http://www.ibm.com/wps/c2a"
  xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
  xmlns:tns="http://www.ibm.com/wps/c2a/CooperativePortlet"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:wssem="http://www.w3.org/ns/sawsdl"
  xmlns:OrderOnt="http://localhost:8080/thesis/Order.owl">
  <types>
    <xsd:schema targetNamespace="http://www.ibm.com/wps/c2a/CooperativePortlet"/>
  </types>
  <message name="customer_name">
    <part name="customer_name" type="xsd:string"
    wssem:modelReference="OrderOnt#Customer_Name"/>
  </message>
  <message name="customer_id">
    <part name="customer_id" type="xsd:string"
    wssem:modelReference="OrderOnt#Customer_ID"/>
  </message>
  <portType name="CooperativePortlet_Service">
    <operation name="customer.details">
      <input message="tns:customer_id"/>
    </operation>
    <operation name="customer.details.by.name">
      <input message="tns:customer_name"/>
    </operation>
  </portType>
  <binding name="CooperativePortletbinding" type="tns:CooperativePortlet_Service">
    <portlet:binding/>
    <operation name="customer.details">
      <portlet:action activeOnStartup="true" caption="customer.details"
        description="Get detail for customer with specified customer id"
        name="customer.details" selectOnMultipleMatch="false" type="standard"
        actionNameParameter="ACTION_NAME"/>
      <input>
        <portlet:param boundTo="request-parameter" caption="customer_id"
          name="customer_id" partname="customer_id"/>
      </input>
    </operation>
    <operation name="customer.details.by.name">
      <portlet:action activeOnStartup="true"
        caption="customer.details.by.name"
        description="Get details for customer with specified name"
        name="customer.details.by.name" selectOnMultipleMatch="false" type="standard"
        actionNameParameter="ACTION_NAME"/>
      <input>
        <portlet:param boundTo="request-parameter"
          caption="customer_name" name="customer_name" partname="customer_name"/>
      </input>
    </operation>
  </binding>
</definitions>
```

CustomerDetail.wsdl



```xml
<?xml version="1.0" encoding="UTF-8"?>
<definitions name="CooperativePortlet"
  targetNamespace="http://www.ibm.com/wps/c2a/CooperativePortlet"
  xmlns="http://schemas.xmlsoap.org/wsdl/"
  xmlns:portlet="http://www.ibm.com/wps/c2a"
  xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
  xmlns:tns="http://www.ibm.com/wps/c2a/CooperativePortlet"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:wssem="http://www.w3.org/ns/sawsdl"
  xmlns:OrderOnt="http://localhost:8080/thesis/Order.owl">
  <types>
    <xsd:schema targetNamespace="http://www.ibm.com/wps/c2a/CooperativePortlet"/>
  </types>
  <message name="order_id">
    <part name="order_id" type="xsd:string"
    wssem:modelReference="OrderOnt#Order_ID"/>
  </message>
  <portType name="CooperativePortlet_Service">
    <operation name="account.details">
      <input message="tns:order_id"/>
    </operation>
  </portType>
  <binding name="CooperativePortletbinding" type="tns:CooperativePortlet_Service">
    <portlet:binding/>
    <operation name="account.details">
      <portlet:action activeOnStartup="true" caption="account.details"
        description="Get account details for specified order id"
        name="account.details" selectOnMultipleMatch="false" type="standard"
        actionNameParameter="ACTION_NAME"/>
      <input>
        <portlet:param boundTo="request-parameter" caption="order_id"
          description="order id for retrieving account details"
          name="order_id" partname="order_id"/>
      </input>
    </operation>
  </binding>
</definitions>
```

AccountDetail.wsdl

APPENDIX C

```xml
<?xml version="1.0" encoding="UTF-8"?>
<definitions name="CooperativePortlet"
  targetNamespace="http://www.ibm.com/wps/c2a/CooperativePortlet"
  xmlns="http://schemas.xmlsoap.org/wsdl/"
  xmlns:portlet="http://www.ibm.com/wps/c2a"
  xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
  xmlns:tns="http://www.ibm.com/wps/c2a/CooperativePortlet"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:wssem="http://www.w3.org/ns/sawsdl"
  xmlns:DisplayOnt="http://localhost:8080/thesis/Display.owl">
  <types>
    <xsd:schema targetNamespace="http://www.ibm.com/wps/c2a/CooperativePortlet"/>
  </types>
  <message name="UI">
      <part name="SWT" type="xsd:string"
        wssem:modelReference="DisplayOnt#SWT"/>
      <part name="Montior" type="xsd:string"
        wssem:modelReference="DisplayOnt#Monitor,
            DisplayOnt#Projector"/>
  </message>
  <portType name="CooperativePortlet_Service"
    wssem:modelReference="DisplayOnt#UI">
    <operation name="Interface">
      <output message="tns:UI"/>
    </operation>
  </portType>
  <binding name="CooperativePortletbinding" type="tns:CooperativePortlet_Service">
    <portlet:binding/>
    <operation name="Interface">
      <portlet:action activeOnStartup="true" caption="" description=""
        name="Interface" selectOnMultipleMatch="false" type="standard"/>
      <output>
        <portlet:param boundTo="request-attribute"
            name="Montior" partname="Montior"/>
        <portlet:param boundTo="request-attribute"
            name="SWT" partname="SWT"/>
      </output>
    </operation>
  </binding>
</definitions>
```

Interface.wsdl

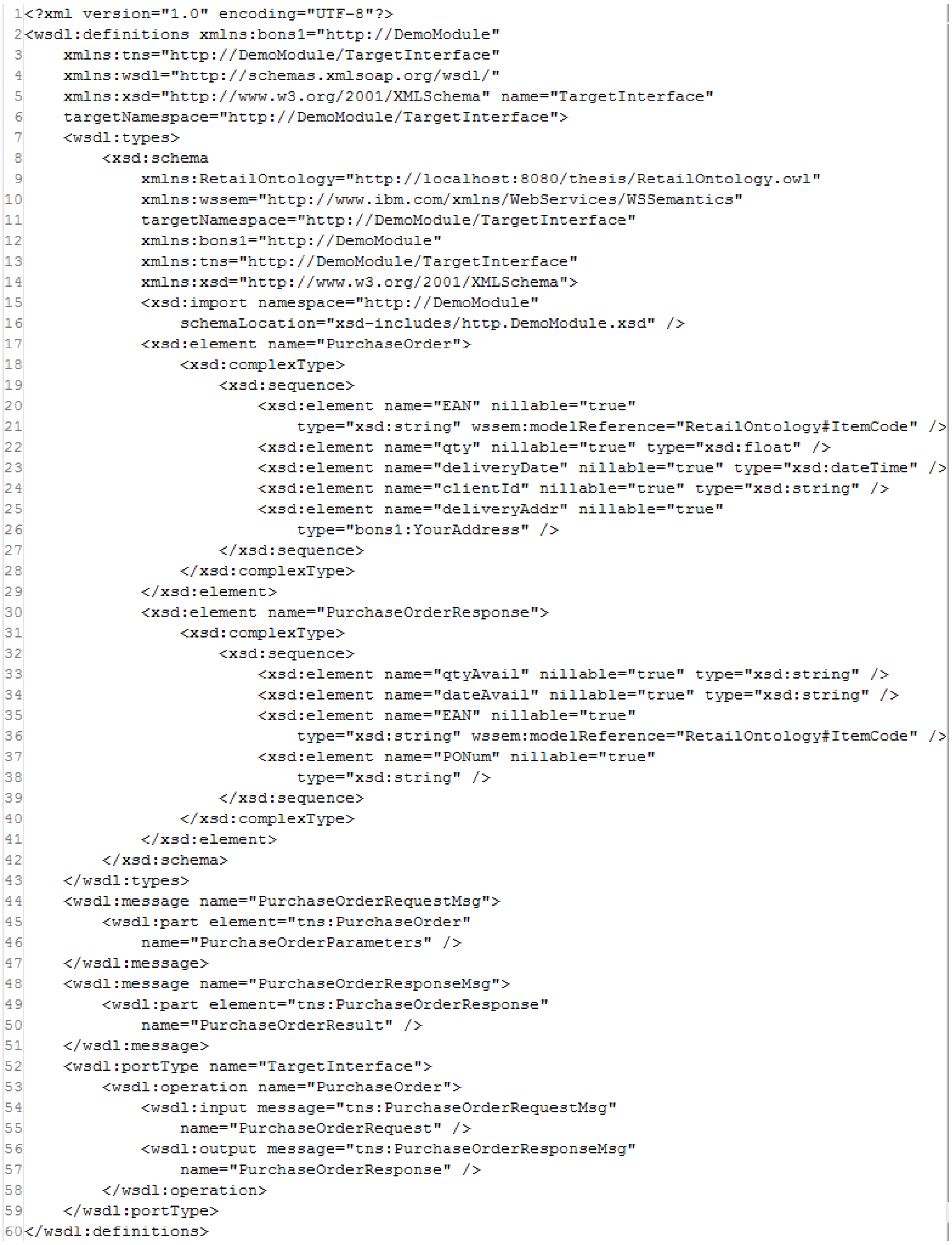
APPENDIX D

```xml
<?xml version="1.0" encoding="UTF-8"?>
<wsdl:definitions xmlns:bons1="http://DemoModule"
    xmlns:tns="http://DemoModule/SourceInterface"
    xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema" name="SourceInterface"
    targetNamespace="http://DemoModule/SourceInterface">
    <wsdl:types>
        <xsd:schema
            xmlns:RetailOntology="http://localhost:8080/thesis/RetailOntology.owl"
            xmlns:wssem="http://www.ibm.com/xmlns/WebServices/WSSemantics"
            targetNamespace="http://DemoModule/SourceInterface"
            xmlns:bons1="http://DemoModule" xmlns:tns="http://DemoModule/SourceInterface"
            xmlns:xsd="http://www.w3.org/2001/XMLSchema">
            <xsd:import namespace="http://DemoModule"
                schemaLocation="xsd-includes/http.DemoModule.xsd" />
            <xsd:element name="Order">
                <xsd:complexType>
                    <xsd:sequence>
                        <xsd:element name="amount" nillable="true" type="xsd:float" />
                        <xsd:element name="UPC" nillable="true"
                            type="xsd:string" wssem:modelReference="RetailOntology#UPC" />
                        <xsd:element name="dueDate" nillable="true" type="xsd:dateTime" />
                        <xsd:element name="acctID" nillable="true" type="xsd:string" />
                        <xsd:element name="deliveryAddr" nillable="true"
                            type="bons1:MyAddress" />
                        <xsd:element name="clientName" nillable="true" type="xsd:string" />
                    </xsd:sequence>
                </xsd:complexType>
            </xsd:element>
            <xsd:element name="OrderResponse">
                <xsd:complexType>
                    <xsd:sequence>
                        <xsd:element name="availableAmount" nillable="true"
                            type="xsd:string" />
                        <xsd:element name="availableItemCode"nillable="true" type="xsd:string"
                            wssem:modelReference="RetailOntology#ItemCode" />
                        <xsd:element name="expectedDueDate" nillable="true"
                            type="xsd:dateTime" />
                        <xsd:element name="confirmationNum" nillable="true"
                            type="xsd:string" />
                    </xsd:sequence>
                </xsd:complexType>
            </xsd:element>
        </xsd:schema>
    </wsdl:types>
    <wsdl:message name="OrderRequestMsg">
        <wsdl:part element="tns:Order" name="OrderParameters" />
    </wsdl:message>
    <wsdl:message name="OrderResponseMsg">
        <wsdl:part element="tns:OrderResponse" name="OrderResult" />
    </wsdl:message>
    <wsdl:portType name="SourceInterface">
        <wsdl:operation name="Order">
            <wsdl:input message="tns:OrderRequestMsg" name="OrderRequest" />
            <wsdl:output message="tns:OrderResponseMsg" name="OrderResponse" />
        </wsdl:operation>
    </wsdl:portType>
</wsdl:definitions>
```

SourceInterface.wsdl

```xml
<?xml version="1.0" encoding="UTF-8"?>
<wsdl:definitions xmlns:bons1="http://DemoModule"
    xmlns:tns="http://DemoModule/SourceInterfaceV1"
    xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema" name="SourceInterfaceV1"
    targetNamespace="http://DemoModule/SourceInterfaceV1">
    <wsdl:types>
        <xsd:schema
            xmlns:RetailOntology="http://localhost:8080/thesis/RetailOntology.owl"
            xmlns:wssem="http://www.ibm.com/xmlns/WebServices/WSSemantics"
            targetNamespace="http://DemoModule/SourceInterfaceV1"
            xmlns:bons1="http://DemoModule"
            xmlns:tns="http://DemoModule/SourceInterfaceV1"
            xmlns:xsd="http://www.w3.org/2001/XMLSchema">
            <xsd:import namespace="http://DemoModule"
                schemaLocation="xsd-includes/http.DemoModule.xsd" />
            <xsd:element name="Order">
                <xsd:complexType>
                    <xsd:sequence>
                        <xsd:element name="amount" nillable="true" type="xsd:float" />
                        <xsd:element name="UPC" nillable="true" type="xsd:string" />
                        <xsd:element name="dueDate" nillable="true" type="xsd:dateTime" />
                        <xsd:element name="acctID" nillable="true" type="xsd:string" />
                        <xsd:element name="deliveryAddr" nillable="true"
                            type="bons1:MyAddress" />
                        <xsd:element name="clientName" nillable="true" type="xsd:string" />
                    </xsd:sequence>
                </xsd:complexType>
            </xsd:element>
            <xsd:element name="OrderResponse">
                <xsd:complexType>
                    <xsd:sequence>
                        <xsd:element name="availableAmount" nillable="true"
                            type="xsd:string" />
                        <xsd:element name="availableItemCode" nillable="true"
                            type="xsd:string" wssem:modelReference="RetailOntology#ItemCode" />
                        <xsd:element name="expectedDueDate" nillable="true"
                            type="xsd:dateTime" />
                        <xsd:element name="confirmationNum" nillable="true"
                            type="xsd:string" />
                    </xsd:sequence>
                </xsd:complexType>
            </xsd:element>
        </xsd:schema>
    </wsdl:types>
    <wsdl:message name="OrderRequestMsg">
        <wsdl:part element="tns:Order" name="OrderParameters" />
    </wsdl:message>
    <wsdl:message name="OrderResponseMsg">
        <wsdl:part element="tns:OrderResponse" name="OrderResult" />
    </wsdl:message>
    <wsdl:portType name="SourceInterfaceV1">
        <wsdl:operation name="Order">
            <wsdl:input message="tns:OrderRequestMsg"
                name="OrderRequest" />
            <wsdl:output message="tns:OrderResponseMsg"
                name="OrderResponse" />
        </wsdl:operation>
    </wsdl:portType>
</wsdl:definitions>
```

SourceInterfaceV1.wsdl



```xml
<?xml version="1.0" encoding="UTF-8"?>
<wsdl:definitions xmlns:bons1="http://DemoModule"
    xmlns:tns="http://DemoModule/TargetInterface"
    xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema" name="TargetInterface"
    targetNamespace="http://DemoModule/TargetInterface">
    <wsdl:types>
        <xsd:schema
            xmlns:RetailOntology="http://localhost:8080/thesis/RetailOntology.owl"
            xmlns:wssem="http://www.ibm.com/xmlns/WebServices/WSSemantics"
            targetNamespace="http://DemoModule/TargetInterface"
            xmlns:bons1="http://DemoModule"
            xmlns:tns="http://DemoModule/TargetInterface"
            xmlns:xsd="http://www.w3.org/2001/XMLSchema">
            <xsd:import namespace="http://DemoModule"
                schemaLocation="xsd-includes/http.DemoModule.xsd" />
            <xsd:element name="PurchaseOrder">
                <xsd:complexType>
                    <xsd:sequence>
                        <xsd:element name="EAN" nillable="true"
                            type="xsd:string" wssem:modelReference="RetailOntology#ItemCode" />
                        <xsd:element name="qty" nillable="true" type="xsd:float" />
                        <xsd:element name="deliveryDate" nillable="true" type="xsd:dateTime" />
                        <xsd:element name="clientId" nillable="true" type="xsd:string" />
                        <xsd:element name="deliveryAddr" nillable="true"
                            type="bons1:YourAddress" />
                    </xsd:sequence>
                </xsd:complexType>
            </xsd:element>
            <xsd:element name="PurchaseOrderResponse">
                <xsd:complexType>
                    <xsd:sequence>
                        <xsd:element name="qtyAvail" nillable="true" type="xsd:string" />
                        <xsd:element name="dateAvail" nillable="true" type="xsd:string" />
                        <xsd:element name="EAN" nillable="true"
                            type="xsd:string" wssem:modelReference="RetailOntology#ItemCode" />
                        <xsd:element name="PONum" nillable="true"
                            type="xsd:string" />
                    </xsd:sequence>
                </xsd:complexType>
            </xsd:element>
        </xsd:schema>
    </wsdl:types>
    <wsdl:message name="PurchaseOrderRequestMsg">
        <wsdl:part element="tns:PurchaseOrder"
            name="PurchaseOrderParameters" />
    </wsdl:message>
    <wsdl:message name="PurchaseOrderResponseMsg">
        <wsdl:part element="tns:PurchaseOrderResponse"
            name="PurchaseOrderResult" />
    </wsdl:message>
    <wsdl:portType name="TargetInterface">
        <wsdl:operation name="PurchaseOrder">
            <wsdl:input message="tns:PurchaseOrderRequestMsg"
                name="PurchaseOrderRequest" />
            <wsdl:output message="tns:PurchaseOrderResponseMsg"
                name="PurchaseOrderResponse" />
        </wsdl:operation>
    </wsdl:portType>
</wsdl:definitions>
```

TargetInterface.wsdl

```xml
<?xml version="1.0" encoding="UTF-8"?>
<wsdl:definitions xmlns:bons1="http://DemoModule"
    xmlns:tns="http://DemoModule/TargetInterfaceV1"
    xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema" name="TargetInterfaceV1"
    targetNamespace="http://DemoModule/TargetInterfaceV1">
    <wsdl:types>
        <xsd:schema
            xmlns:RetailOntology="http://localhost:8080/thesis/RetailOntology.owl"
            xmlns:wssem="http://www.ibm.com/xmlns/WebServices/WSSemantics"
            targetNamespace="http://DemoModule/TargetInterfaceV1"
            xmlns:bons1="http://DemoModule"
            xmlns:tns="http://DemoModule/TargetInterfaceV1"
            xmlns:xsd="http://www.w3.org/2001/XMLSchema">
            <xsd:import namespace="http://DemoModule"
                schemaLocation="xsd-includes/http.DemoModule.xsd" />
            <xsd:element name="PurchaseOrder">
                <xsd:complexType>
                    <xsd:sequence>
                        <xsd:element name="EAN" nillable="true"type="xsd:string" />
                        <xsd:element name="qty" nillable="true" type="xsd:float" />
                        <xsd:element name="deliveryDate" nillable="true" type="xsd:dateTime" />
                        <xsd:element name="clientId" nillable="true" type="xsd:string" />
                        <xsd:element name="addr" nillable="true" type="bons1:YourAddress" />
                    </xsd:sequence>
                </xsd:complexType>
            </xsd:element>
            <xsd:element name="PurchaseOrderResponse">
                <xsd:complexType>
                    <xsd:sequence>
                        <xsd:element name="qtyAvail" nillable="true" type="xsd:string" />
                        <xsd:element name="dateAvail" nillable="true" type="xsd:string" />
                        <xsd:element name="EAN" nillable="true" type="xsd:string" />
                        <xsd:element name="PONum" nillable="true" type="xsd:string" />
                    </xsd:sequence>
                </xsd:complexType>
            </xsd:element>
        </xsd:schema>
    </wsdl:types>
    <wsdl:message name="PurchaseOrderRequestMsg">
        <wsdl:part element="tns:PurchaseOrder"
            name="PurchaseOrderParameters" />
    </wsdl:message>
    <wsdl:message name="PurchaseOrderResponseMsg">
        <wsdl:part element="tns:PurchaseOrderResponse"
            name="PurchaseOrderResult" />
    </wsdl:message>
    <wsdl:portType name="TargetInterfaceV1">
        <wsdl:operation name="PurchaseOrder">
            <wsdl:input message="tns:PurchaseOrderRequestMsg"
                name="PurchaseOrderRequest" />
            <wsdl:output message="tns:PurchaseOrderResponseMsg"
                name="PurchaseOrderResponse" />
        </wsdl:operation>
    </wsdl:portType>
</wsdl:definitions>
```

TargetInterfaceV1.wsdl

WORKS CITED

1 The WSDL standard is defined by the W3C at http://www.w3.org/TR/wsdl.

2 The XML standard is defined by the W3C at http://www.w3.org/XML.

3 He, J. & Yen, I (2007). Adaptive User Interface Generation for Web Services. *Proceedings from the IEEE International Conference on e-Business Engineering*, 536 – 539.

4 Portlets as implemented in the Java programming language are defined by JSR 168, http://jcp.org/en/jsr/detail?id=168.

5 http://www.eclipse.org.

6 OSGi originally stood for Open Service Gateway Initiative. This term is no longer used and the alliance is now known simply as OSGi. http://www.osgi.org.

7 http://www.ibm.com/software/lotus/products/expeditor/.

8 The XML Schema standard is defined by the W3C at http://www.w3.org/XML/Schema.

9 http://www.w3.org/2001/sw/.

10 RDF is defined by the W3C at http://www.w3.org/TR/rdf-syntax-grammar/.

11 OWL is defined by the W3C at http://www.w3.org/2004/OWL/.

12 Dong, X., Halevy, A. Y., Madhavan, J., Nemes, E., & Zhang, J. (2004) Similarity Search for Web Services. *Proceedings from the Very Large Databases conference*.

13 http://www.w3.org/2002/ws/sawsdl/.

14 http://www.w3.org/TR/sawsdl/.

15 Verma, K., Sheth, A. (2007) Semantically Annotating a Web Service, *Internet Computing, IEEE Publication*, *March-April*, 83-85.

16 Syeda-Mahmood, T., Shah, G., Akkiraju, R. Ivan, A., & Goodwin, R. (2005). Searching Service Repositories by Combining Semantic and Ontological Matching. *Third International Conference on Web Services (ICWS).*

17 http://pipes.yahoo.com/pipes/.

18 http://softwarecommunity.intel.com/articles/eng/1461.htm.

19 Murthy, S., Maier, D., & Delcambre, L. (2006). Mash-o-matic, *Proceedings of the 2006 ACM Symposium on Document Engineering*, 205-214.

20 http://www.w3.org/TR/xpath.

21 Comella-Dorda, S., Wallnau, K., Seacord, R., & Robert, J. (2000). A Survey of Black-Box Modernization Approaches for Information Systems. *16th IEEE International Conference on Software Maintenance (ICSM'00)*, 173

22 http://www.google.com/codesearch.

23 Miller, G. (1983) WordNet: A lexical database for the English language. *Communications of the ACM*.

24 Phifer, G. (2005). Portals Provide a Fast Track to SOA. *Business Integration Journal*, Nov/Dec.

25 Knublauch, H. Fergerson, R., Noy, N., & Musen, M. (2004). The Protégé OWL Plugin: An Open Development Environment for Semantic Web Applications. http://protege.stanford.edu/plugins/owl/publications/ISWC2004-protege-owl.pdf.

26 http://www.jiwire.com/.

27 http://www.ibm.com/developerworks/lotus/library/expeditor-toolkit/.

28 Knublauch, H. (2004) Ontology-Driven Software Development in the Context of the Semantic Web: An Example Scenario with Protégé/OWL. *International Workshop on the Model-Driven Semantic Web*.

29 Roy-Chowdhury, A. & Wu, Y. (2005) Developing JSR 168 compliant cooperative portlets. http://www.ibm.com/developerworks/websphere/library/techarticles/0412_roy/0412_roy.html

30 http://www.alphaworks.ibm.com/tech/wssem.

31 http://www.uml.org/.

32 Evans, A. (1998). Reasoning with UML Class Diagrams. *Proceedings from the Workshop Industrial-Strength Formal Specification Techniques*.

33 Gasevic, D., djuric, D., Devedzic, V., & Damjanovic, V. (2004). From UML to ready-to-use OWL ontologies. *Proceedings from the 2nd International IEEE conference on Intelligent Systems*, 485 – 490.