METHOD AND MODELS TO ENABLE OPTIMAL AUTOMATED SERVICE COMPOSITION

by

John D. McDowall
A Dissertation
Submitted to the
Graduate Faculty
of
George Mason University
in Partial Fulfillment of
The Requirements for the Degree
of
Doctor of Philosophy
Information Technology

Committee:

_____	Dr. Larry Kerschberg, Dissertation Co-Director
_____	Dr. Alexander Brodsky, Dissertation Co-Director
_____	Dr. Sam Malek, Committee Member
_____	Dr. Stephen Nash, Senior Associate Dean
_____	Dr. Kenneth S. Ball, Dean, Volgenau School of Engineering
Date: _____	Spring Semester 2015 George Mason University Fairfax, VA

Method and Models to Enable Optimal Automated Service Composition

A Dissertation submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy at George Mason University

by

John D. McDowall
Master of Science
Boston University, 1999
Bachelor of Science
United States Naval Academy, 1989

Co-Directors: Larry Kerschberg, Professor, and Alexander Brodsky, Associate Professor
Department of Computer Science

Spring Semester 2015
George Mason University
Fairfax, VA

DEDICATION

This is dedicated to the glory of God and to my parents, Adele and Jim, who encouraged my curiosity. Also to my wife Michele and our children Lindsay and Brooke, who have learned to endure the results of my parents' encouragement.

ACKNOWLEDGEMENTS

I would like to thank the members of my committee, especially co-directors Larry Kerschberg and Alexander Brodsky, for countless hours reviewing and editing my content. I would also like to thank Sam Malek for his valuable guidance and very flexible schedule, and the late Anderw Sage, an original member of my committee whose work on complex adaptive systems was a major influence on my thinking. Finally, I would like to thank the core members of my dissertation writers' group: Susan Farley, Mark Coletti, and Jeff Bassett. Their encouragement, input, and unflagging optimism were invaluable.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF DEFINITIONS

LIST OF ABBREVIATIONS OR SYMBOLS

Application Programming Interface ..... API
Business Process Execution Language ..... BPEL
Business Process Execution Language for Web Services BPEL4WS
Business Process Modeling Ontology ..... BPMN
Cluster-Optimizing-Diversity ..... COD
Composite Alternative Recommendation Development ..... CARD
DARPA Agent Markup Language ..... DAML
DARPA Agent Markup Language for Services ..... DAML-S
Decision Guidance Structured Query Language ..... DG-SQL
Defense Advanced Research Projects Agency ..... DARPA
Description Logics ..... DL
Extensible Markup Language ..... XML
Foundation for Independent Physical Agents ..... FIPA
Graphical User Interface ..... GUI
Managing End-To-End Operations—Semantic Web Services ..... METEOR-S
Model-Driven Architecture ..... MDA
Optimization Programming Language ..... OPL
OWL for Services ..... OWL-S
Public Key Infrastructure ..... PKI
Quality of Service ..... QoS
Representational State Transfer ..... REST
Resource Description Framework ..... RDF
Self-Architecting Software Systems ..... SASSY
Semantically Annotated WSDL ..... SAWSDL
Service Oriented Architecture ..... SOA
Short Message Service ..... SMS
Simple Object Access Protocol ..... SOAP
Structured Query Language ..... SQL
Support Vector Machine ..... SVM
Universal Modeling Language ..... UML
Universal Resource Identifier ..... URI
Universal Transverse Mercator ..... UTM
Virtual Service Instance ..... VSI
Web Application Description Language ..... WADL
Web Ontology Language ..... OWL
Web Services BPEL ..... WSBPEL

Web Service Description Language..... WSDL
Web Service Description Language-Semantic .....WSDL-S
Web Services Modeling Framework .....WSMF
Web Services Modeling Ontology..... WSMO
World Wide Web Consortium W3C
XML Metadata Interchange.....XMI
XML Schema Definition.....XSD
Zone Improvement Plan.....ZIP

ABSTRACT

METHOD AND MODELS TO ENABLE OPTIMAL AUTOMATED SERVICE COMPOSITION

John D. McDowall, Ph.D.

George Mason University, 2015

Dissertation Co-Director: Dr. Larry Kerschberg

Dissertation Co-Director: Dr. Alexander Brodsky

Since the development of the Service Oriented Architecture concept, business analysts and system developers have looked forward to the day when they could reconfigure applications to adapt to new business by combining services in new ways to adapt to changing business needs. Technologies such as the Web Services Description Language and Business Process Modeling Notation (BPMN) provide key building blocks but are not sufficient to enable run-time reconfiguration of services. To enable this functionality, this research develops Druid, a framework for dynamically composing web services into executable processes based on a business process model defined using the BPMN modeling language. To support this framework, this research develops a service description modeling language, extensions to the BPMN language, and a formal model for composing services based on a business model. This research also develops a Quality of Service (QoS) model used for calculating the optimal service composition.

1. INTRODUCTION

The development of the Service Oriented Architecture (SOA) concept in the late 20th century promised a revolution in system development and integration. Despite a great deal of research and commercial development, we have not realized the promise of systems that can dynamically discover and invoke new services. The flexible, loosely-coupled systems envisioned when the SOA concept was first proposed have not materialized. In practice, SOA services are widely deployed, but as a means to allow third-party developers to integrate services into new, value-added applications known as "mashups." With the exception of some research programs, systems that allow end users to compose services from different systems into a new workflow do not exist. We want to field systems that give end users and business analysts the power to define a process they need to choreograph, find services that can contribute to that process, compose the services together into potential executable workflows, and then select the best option from among the candidate compositions, based on quality of service metrics. And we want to do this without the long development-integration-test cycles that have characterized software development for decades.

This research focuses on automating the dynamic composition of services into optimal executable workflows. The approach taken in this dissertation involves an extension to the Business Process Modeling Notation (BPMN) modeling language, the

creation of a semantic service description language, and the specification of an optimization model that selects the optimal composition based on a number of quality-of-service metrics.

1.1. Motivation

The SOA concept has an instinctive appeal, beyond the time and cost savings inherent in reusing existing services, rather than developing new components from scratch. Part of this appeal comes from the fact that the SOA concept in information systems parallels the real world as we experience it: we live in a service-oriented world. Consider the following scenarios:

- A man's car is in need of a tune-up, and rather than go to the trouble of doing the maintenance himself, he drops his car off with a mechanic who changes the oil and sparkplugs, lubricates the chassis, and checks the brakes.
- A woman wants to ensure she invests her retirement savings in a way that will protect her principal while ensuring a reasonable rate of return. Rather than devoting many hours to monitoring the financial markets and updating her portfolio periodically, she contracts with a financial advisor to manage her retirement savings.
- A family is planning a vacation, and rather than find a listing of all the hotels near their destination and contacting each one individually to determine the cost and availability of rooms, they contact a travel agent

with their destination, dates of travel, and budget and ask the travel agent to make the arrangements.

Each of these scenarios is very different from the others, but they share a common theme: instead of doing the work himself, a person contracts with a service provider to perform the desired work. Each service provider has some interface with some required inputs and some expected outputs (e.g., the mechanic requires in-person interaction, with a car in need of maintenance as the input and a properly maintained car as the output).

Economists describe this outsourcing of tasks as specialization, where someone with advanced training or experience in the matter at hand, can perform the task better than a layman. In addition to having the advanced skills required to perform the task well, the service provider frees up the consumer to spend time on other tasks. The development of the SOA concept extends this idea from the physical world into the digital world of information systems. Instead of a system developer creating each and every component of a system, some functions are outsourced to external service providers.

We live in a service-oriented world; SOA merely extends this idea to the way we design and implement information systems. However, current technologies have several limitations that prevent us from realizing the full potential of SOA systems; these limitations are discussed in the following section.

1.2.Research Gap [Related Work and its Limitations]

The initial development of the SOA concept (1) was based on web services described using the Web Service Description Language (WSDL) (2,3) and exposed a Simple Object Access Protocol (SOAP) interface to service requestors. The WSDL

specification includes only syntactic information—names and data types of individual elements. However, the problem with purely syntactic descriptions is that they make matchmaking difficult. Each WSDL document describes the operations of a given service and each operation's inputs and outputs. With a syntactic description, parameter names (operation names, input and output names) must be identical or the matchmaking process will be unsuccessful. For example, a parameter called "zipCode" will not match a parameter with the same meaning named "zip_code." Coordinating operation and parameter names across a large number of service providers is impractical, but adding semantic annotations to a service description offers a clean solution to this problem.

To address the limitations of syntax-based service description models, several research efforts have developed semantic service specifications. The WSDL-Semantic (WSDL-S) project (4) extended the WSDL specification with semantic markup of the operations and parameters for services, as did the Semantically Annotated WSDL (SAWSDL) project (5), which added semantic annotations to both WSDL and XML schema to address the difficulties of syntactic matchmaking. This was accomplished by adding a reference to an external ontology, as well as optional generalization or specialization notations. Each of these enabled semantic matchmaking techniques, greatly improving the ability to dynamically compose services. The Web Ontology Language (OWL) for Services (OWL-S) project (6) took a different approach by creating a process description using OWL and grounding it in WSDL. OWL-S also captured the intended business process in the OWL-S description by annotating the individual operations to indicate how they might be composed together. In addition to enabling semantic

matchmaking, the OWL-S approach brought to bear the power of the OWL specification's support for machine reasoning about the ontology specifications. All of these projects built upon the WSDL specification, adding semantics to the basic WSDL structure in order to enable service matchmaking. However, by basing their work on WSDL, these projects limited themselves to SOAP-based web services.

Meanwhile a new style of web services, based on the Representational State Transfer (REST) model (7) was gaining popularity; because the WSDL specification did not initially support REST services, none of the previously-described approaches to web service semantics could accommodate REST services. However, the original REST service proposal did not include a specification for machine-readable service descriptions. While the release of WSDL 2.0 included support for REST services, WSDL has not been widely adopted as a means of describing REST service interfaces. The Web Application Description Language (WADL) (8) was developed as a WSDL analog to specifically support REST services, but like WSDL, it includes no support for semantic description of the service operations or parameters. The WADL specification has not been widely adopted, nor does it satisfy our need to semantically describe service operations or parameters.

In addition to those limitations already mentioned, all of these service description formats share an additional limitation: they describe only web services. That is, they lack the ability to describe any service other than those offered via an interface to a digital information system. In practice, there are many services that are either purely physical (e.g., a doctor interpreting an X-ray) or may be offered in both physical and digital forms.

For example, a hotel may offer a web service for completing a reservation and also maintain a staff that allows customers to make reservations over the telephone. Both are equivalent services with different interfaces, but we have no standardized way to describe the physical service. This is important because most practical business processes require some level of interaction with the physical world, from something as simple as a final approval of an expense report, to complex services such as imagery analysis. Any system or framework to compose services into useable workflows must account for the possibility of physical services as part of the composition.

However, none of the previous service description efforts developed a service description that encompassed both SOAP- and REST-based services, and none of them developed a general service description language suitable for describing both physical and digital services. Moreover, none of these service description efforts included a means for describing how those services could be incorporated into a larger process. A service description language is of limited utility by itself. The necessary complement to service descriptions is a means of specifying a process the user wishes to implement. Services are rarely used individually; they are most often part of a larger business process. A business process is a sequential series of discrete activities intended to accomplish some meaningful unit of work. To make dynamic service composition useful, a system requires some means for specifying the overall business process the services support. A convenient way of doing this would be to leverage process modeling languages such as Business Process Modeling Language (BPMN) (9,10) to define a process and then match services to that process. This would allow business analysts to express their needs in a

familiar language, minimizing the ambiguity inherent in communicating their needs to software developers.

One approach to defining a process model for composing web services was explored by the Web Services Modeling Ontology (WSMO) (11) and the Web Services Modeling Framework (12). Each approach developed a framework for describing web services and business processes in a manner that enabled matchmaking between individual steps in the process and available services as well as matchmaking among service input and output parameters. While these systems both demonstrated good results, they were limited to SOAP-based web services and did not use a standardized process modeling language such as BPMN. Instead, they each implemented their own process definition notation to express the desired process flow.

Another research effort that sought to define a process specification for service composition was the METEOR-S (Managing End-to-End Operations – Semantic Web Services) project (13,14). METEOR-S used Universal Modeling Language (UML) activity diagrams to define business processes, together with a semantic web service description, to enable the automated matchmaking of services to process steps and to each other. The METEOR-S approach worked well for UML diagrams, but did have some limitations. Like other approaches, it was limited to SOAP-based web services. In addition, UML is a visual modeling language intended for human consumption and interpretation; it has no representation designed to facilitate automated processing. While the layout of a UML model can be described using XML Metadata Interchange (XMI) specification (15), XMI is suited to exchanging UML models among modeling tools, not

for reasoning about the contents of the model. Also, UML diagrams are inherently more ambiguous than specialized process modeling languages such as BPMN because they lack some of the fine-grained flow-control notations that are part of BPMN.

Taking a different approach to the problem of mapping services into a process model, the SAWSDL-MX (16,17) and the OWLS-MX (18–21) families of matchmakers developed *semantic matchmakers* that used the SAWSDL and OWL-S specifications, respectively, to perform matchmaking among service interfaces. Neither project specifically tried to match services to steps in a process flow, but instead tried to determine if two service interface definitions were semantically equivalent. This is, in effect, matching a service description to a process step that describes the type of service needed (i.e., the process defines a series of service templates required to complete the process). While they both demonstrated success, each of these approaches was limited to using SOA-based web services within their own service description language, and did not explore specific process modeling languages.

What all these process-definition and service-composition methods lack is a means for assessing alternative compositions based on Quality of Service (QoS) metrics. Given some number of potential service compositions, currently there is no means of determining which service composition is the best available. This assessment should be based on an analysis of QoS metrics of each of the services and the QoS of the composition as a whole. In addition, little work has been done on documenting the QoS attributes of services beyond the network-focused criteria of response times and data throughput as in M. Alasti et al(22,23). These criteria are appropriate to web services

delivering streaming video, but are not well suited to QoS aspects such as cost minimization or user satisfaction.

One approach to assessing QoS aspects of compositions is the Self-Architecting Software Systems (SASSY) project (24). However, SASSY has focused on assembling a software architecture based on the QoS attributes of architecture components and the desired QoS expressed in an architecture model, and not on assessing the QoS of individual services. While SASSY demonstrated a successful architecture composition on this basis, it was designed for selecting software components based on the QoS of those pre-defined components; SASSY was not designed to support the run-time discovery and composition of general purpose services.

However, to assess the QoS of alternative service compositions we need to provide users with a flexible means of defining and evaluating service QoS. Relatively little work has been done on the optimization of service compositions. The work described by Mabrouk et al (25–27) built a semantic QoS model that could be used to assess service compositions based on the evaluation of QoS attributes, but their work focused on documenting the QoS for services and did not extend to optimizing the composition of a set of services assembled to complete a workflow. In fact, as described by Yu and Lin in (28), such an optimization is an NP-hard problem that has received little attention.

1.3.Thesis and Contributions

The focus of this dissertation is to address the limitations described above and explain the development of tools and models that overcome those limitations.

Thesis

It is possible to develop a system that accepts as input a semantically-annotated process model, parses that model and finds candidate services that perform each of the activities described in the model, and calculates an optimal composition of the available services based on QoS attributes of the services and the overall composed process. Furthermore, it is possible to develop a service description language that enables the activity-to-service and service-to-service matchmaking necessary to enable the service composition. Finally, it is possible to optimize the selection of services based on QoS factors defined for each service and for the service composition as a whole.

Contributions

Composing services into a complete workflow traditionally has been a manual programming process that cannot be automated using current techniques. To enable this automation, we need to match services to each activity in a process model; we need to be able to match services to each other to create compositions, and, more importantly, we need to select an *optimal* composition. To achieve these goals, this research develops a system that allows, as an example, business users to automate the execution of a business process such as booking travel reservations. To enable activity-to-service matchmaking, this research defines 1) an extension to the BPMN language for the semantic annotation of individual process activities and 2) a language for specifying services that enables both automated activity-to-service matchmaking and service-to-service matchmaking. To select the optimal service composition, this research develops several elements. The first is a model that describes the QoS metrics of services and processes. The QoS model is combined with a formal definition of processes and services to develop an optimization

model. This optimization model enables the selection of an optimal service composition from among multiple candidate compositions. To demonstrate the feasibility of this work, this research also develops a proof of concept prototype that implements all of these developments.

More specifically, the key contributions of this research are summarized as follows:

- **Service Composition Methodology** that provides a step-by-step explanation of the process of composing services into an optimal workflow. This process starts with a BPMN model and continues through the selection of an optimal service composition.
- **Semantic Extension to BPMN** to support the semantic annotation of process activities. This model defines the business process that the service composition will perform.
- **Service Description Language** supporting the semantic description of digital and physical services. This language enables the automated matching of services to process activities.
- **QoS Model** that describes QoS characteristics of digital and physical services. This model defines the metrics that will be used in the optimization calculation.
- **Optimal Service Composition Model** that formally defines an optimal service composition. This is a formal mathematical definition of the service composition process and the optimization calculation.

- **QoS Optimization Implementation** in Optimization Programming Language (OPL) using the IBM CPLEX suite. This is an implementation of the optimization model using mathematical programming..
- **Proof of Concept Prototype** demonstrating the feasibility of this approach. This is a collection of tools and components that implement the processes and specifications defined in this research.

The design goals of a system that supports the type of dynamic service composition described above include allowing a service provider to describe services with the requisite semantic detail; this requires a service description language that supports the semantic annotation of the operations and the inputs and outputs of both digital and physical services. The system should also allow a business user to specify the desired process using a standard process modeling language; this requires the extension of a process modeling language to support the semantic annotation needed for activity-to-service matchmaking. Finally, the system should be able to analyze the QoS characteristics of the services and develop a recommendation for the optimal service composition based on those QoS metrics; this requires a formal model of services, processes, and QoS metrics that can be subjected to optimization analysis.

The key contributions of this research are described in the following paragraphs:

Semantic Extensions to BPMN

Activity-to-service matchmaking requires that activities in a process model include a semantic specification of the task performed by each activity. In order to enable the semantic specification of individual activities, this research extends the BPMN

language to support the annotation of individual process steps with semantic descriptions that reference external ontologies describing the relevant business domain. These references enable an analyst to specify the type of task performed at each step of the process, and optionally the semantic types of the inputs and outputs to that step in the process. For example, an activity named "Get Weather" may be assigned a task type "getCurrentWeather" with a single input parameter with a semantic type "postal_code" and two output parameters with semantic types "max_temp" and "current_temp."

Service Description Language (Ogma[1])

Activity-to-service matchmaking also requires that service descriptions include semantic detail about the operations, inputs, and outputs of that service. This research develops the service description language Ogma to represent the information required to effect automated matching of process steps to services. The Ogma language enables the semantic description of each type of task performed by the service as well as the service inputs and outputs. The language also include information such as how to invoke the service (i.e., binding information) and other information necessary to match services to process activities and to compose services with each other. The Ogma description is specified using OWL to enable automated reasoning about individual services, such as class/subclass (i.e., "is a") relationships and equivalence assessment based on common parameters and effects.

QoS Model (Sucellos[2])

[1] Ogma: a Celtic god of language
[2] Sucellos: a Celtic god of time

The selection of an optimal service composition based on QoS metrics requires a QoS model capable of capturing QoS metrics for both web and physical services. To enable the annotation of service descriptions with QoS parameters, this research develops Sucellos, a QoS model that captures metrics such as cost, user rating, and the like that are appropriate . The Sucellos model is specified using OWL to more easily integrate with the Ogma service description model.

Optimal QoS Service Composition Model (Ecne[3])

Selecting the optimal service composition from among multiple candidates requires a means of specifying a business process, services, and the QoS parameters of each composition so the alternative compositions may be analyzed. To ennable this specification, this research develops Ecne, a formal description of processes, services, and QoS parameters that can be subjected to an analysis that calculates the optimal service composition based on the Ecne model.

QoS Optimization Implementation

To calcuatle the optimal service composition based on the Ecne model requires a means of evaluating the alternatives and arriving at a recommendation. To execute the analysis of the Ecne formalism, this research develops an implementation model expressed in the Optimization Programming Language (OPL) and that is executed using IBM's CPLEX environment.

Proof of Concept

[3] Ecne: a Celtic god of wisdom

To demonstrate the feasibility of this approach, a proof of concept prototype system that accepts as input a semantically-annotated process model defined using BPMN and produces as output a QoS-optimized service composition that implements the process model. First, the BPMN model is decomposed into individual activities and each activity is matched to one or more semantic service descriptions. Next, each of the matched services is compared to the other services to determine which services may be composed into candidate workflows that can perform the process described by the model. Then, each of the workflows is assessed by comparing the QoS metrics of each service and the QoS characteristics of the complete workflow, and finally an optimal workflow recommendation is developed based on the QoS information about each service and the overall process. The QoS information about each service and the process conforms to a formal QoS model.

1.4. Dissertation Organization

The remainder of this dissertation is organized as follows. In Chapter 2, I review related work in the fields of service description models, business process modeling languages, and (service) optimization. In Chapter 3, I describe the overall service composition landscape that this research encompasses. In Chapter 4, I describe the Ecne QoS model and the formal definition of the optimization problem. I explain design and implementation of the Ogma service description language. In Chapter 5, I discuss the extension of the BPMN modeling language to express the task type of each activity in a process model and the semantic types of the inputs and outputs of each activity. In Chapter 6, I describe the Ogma service description language used to capture the service

description information needed to enable activity-to-service and service-to-service matchmaking. In Chapter 7, I describe the proof of concept prototype I developed to demonstrate the feasibility of the key contributions of this research. Finally, Chapter 8 summarizes my conclusions and outlines areas for future research.

2. RELATED WORK

Since the development of the SOA concept, a variety of researchers have looked into the problem of service composition. While few researchers have looked at the complete service composition lifecycle from process definition to optimal composition selection, many research projects have focused on different aspects of the problem such as service descriptions or process models. These efforts are described in the sections that follow.

2.1. Service Descriptions

Neither the WSDL nor WADL specifications include the semantic annotations necessary for service matchmaking, nor did either specification include elements for describing QoS aspects. To address these limitations, a variety of research projects have developed extensions or alternatives to the standard service descriptions that capture the desired information.

2.1.1. WSDL-S

The Web Service Description Language-Semantic (WSDL-S) model (4) builds on the WSDL standard, using the extensibility mechanism included in the WSDL specification to add semantic annotations to service descriptions.

The WSDL-S model was developed with five design goals. The first was to build on existing web services standards. Because there is no widely accepted standard for

describing REST services, the WSDL-S model focused on describing SOAP-based services.

The second and third design goals of the WSDL-S model are related. The second goal was that the means of specifying the semantic attributes of a service should be independent of the semantic representation language. The third goal was that the means of semantically annotating the service description should support multiple representations of the same item written in different semantic representation languages.

The fourth design goal was to support the semantic description of data types represented by the XML Schema Definition (XSD) Language (this is in accordance with the first goal of building on existing standards). This goal leverages the practice most web service interfaces employ of describing data inputs and outputs using XML Schema. The final design goal was related to the fourth goal; it aimed to provide support for rich mappings of XSD data types into ontological representations without regard to the language used to define the ontology.

The WSDL-S model extends the basic WSDL model with several attributes that provide Universal Resource Identifiers (URIs) that link WSDL elements to semantic descriptions of those elements. These semantic annotations describe the inputs, outputs, and operations offered by a service, as well as describing the preconditions necessary to invoke any of the operations. There are also semantic annotations that describe the service category.

2.1.2. OWL-S

The Ontology Web Language for Services (OWL-S) project (6) began as the Defense Advanced Research Projects Agency (DARPA) Agent Markup Language for Services (DAML-S) (29–31). DAML-S was developed to describe the services offered by intelligent agents to enable the discovery and composition of those services through the autonomous interaction of those agents. The DARPA Agent Markup Language (DAML) ultimately evolved into OWL, and so DAML-S evolved into OWL-S, retaining the same goals as DAML-S.

The OWL-S project incorporated the semantics into the interface description by creating a new service description format based on OWL. This format includes all the information available in a WSDL document, but created a new description model designed to take advantage of OWL's support for reasoning.

The OWL-S service description model is composed of three main parts: a service profile, a process model, and a grounding. The service profile includes the syntactic and semantic descriptions of the service interface, its operations, and the data each operation consumes and produces. The process model describes how the different operations offered by the service can be invoked in combinations to perform more complex tasks than any of the individual operations can perform. The OWL-S grounding contains the information needed to bind to and invoke the service's individual operations.

While the OWL-S project was focused on providing descriptions for SOAP-based services, the model could theoretically be extended to describe REST services or web services based on any other technology.

2.1.3. SAWSDL

In 2006, the World Wide Web Consortium (W3C) convened a working group to address the semantic shortcomings of the WSDL specification that had been identified by the developers of OWL-S, the Web Services Modeling Ontology (WSMO, described below), and similar projects. This group's final product was the Semantic Annotations for WSDL (SAWSDL) specification (5,32).

While the SAWSDL specification uses different terminology than the WSDL-S specification, both projects employ the same technique for adding semantic information to a basic WSDL document. Each includes references to an externally defined ontology independent of the ontological language employed. Also, each includes mappings between the ontological concepts and XSD types used to define the data input and output parameters.

2.1.4. WADL

In 2006, the Web Application Description Language (WADL) (8) was proposed as a means for formalizing descriptions of REST-based services in a machine-readable format. Since that time, it has not been widely used in practice. Like the WSDL specification, the WADL specification describes the syntax of a service interface but does not capture the semantics necessary to understand the meaning of elements in the interface. The WADL specification does not include the information necessary to enable service matchmaking, nor does it include elements for describing QoS aspects of services.

2.1.5. WSMO

The Web Services Modeling Ontology (WSMO) (11) is a semantic service description model based on the Web Services Modeling Framework (WSMF) (12). Like OWL-S, WSMO was developed as a mechanism to enable intelligent agents to work together to accomplish a set task. WSMO was developed in part to address perceived shortcomings of the OWL-S approach, and like the OWL-S model WSMO created an alternative description format independent of the WSDL standard. The WSMO model is composed of four main elements: ontologies, web services, goals, and mediators.

WSMO ontologies provide domain-specific descriptions of the terms used to describe the other elements of a WSML model. These ontologies provide both a formal description of the service's semantics and a link between the human-readable and machine-readable terminologies.

WSMO web services are conceptually similar to web services described by other service description models, in that they describe pieces of functionality that can be combined in different ways to perform more complex tasks. Still, WSMO service descriptions are different from those found in a WSDL. Within WSMO, a service is described in terms of properties, functionality, and behavior; behavioral descriptions are not part of the WSDL specification and are one of the unique contributions of WSMO.

WSMO goals are defined independently of services. This makes it possible for a user to specify a goal independently of any conception of the services that are available. This independence of descriptions ensures the intelligent agents can compose services to achieve the goal using the most efficient combination of available services.

The WSMO project also defined mediators; mediators are mechanisms for translating between heterogeneous descriptions of different services. Mediators may be applied to ontologies, services, and goals. The purpose of a mediator is to enable interoperability between different WSMO instances that may be based on different assumptions and use different terminologies.

Like OWL-S, the WSMO model was designed to represent SOAP-based web services. It should be possible to represent REST-based services or other web service paradigms within the WSMO model, but it was not designed to represent other service types and its suitability for that purpose has not been demonstrated. One key difference between OWL-S and WSMO is that, while OWL-S is based on OWL-DL (where DL denotes the OWL dialect supporting the computational completeness and decidability of Description Logics), WSMO is based on F-Logic, an alternative language for representing ontologies with Description Logics.

2.2.Process Modeling Languages

Process modeling languages vary widely, from the very simple information flowchart to highly sophisticated modeling languages such as BPMN. To be suitable for defining a business process with sufficient detail to enable automated service matchmaking, composition, and optimization, a business process modeling language must be precise enough specify the types of activities in the business process and the relationships among those activities. Several research efforts that have investigated different options for modeling business processes are discussed below.

2.2.1. BPMN

BPMN is a graphical language designed for use by business analysts in specifying complex business processes. In (33), the conversion of BPMN models to Business Process Execution Language (BPEL) is examined and several conceptual mismatches are identified. These conceptual mismatches make the automated transformation from graphical BPMN to executable BPEL problematic. It should be noted that this work was based on the BPMN 1.2 specification (9). BPMN 1.2 was a purely graphical language, with no XML representation defined in the specification and no means for executing a BPMN 1.2 model.

The 2011 release of BPMN 2.0 incorporated a specification for representing BPMN models in XML. The inclusion of a formal XML syntax makes it easier to transform a BPMN model into other representations, and also makes BPMN natively executable by a suitable execution engine. Vendors such as Oracle and BonitaSoft offer engines that will execute BPMN 2.0 models, making it possible for a business analyst to use BPMN to specify service interactions graphically and execute the resulting service composition. However, model-to-service mappings are not part of the BPMN specification, so each of these vendors implements its own mapping scheme and service mappings are not portable across execution engines from different vendors.

2.2.2. BPEL

The most common composition language in use today is the Business Process Execution Language (BPEL) (34), also known as BPEL for Web Services (BPEL4WS) and Web Services BPEL (WSBPEL). BPEL is an XML-based language that remains under active development and is supported by a number of commercial products.

The BPEL specification defines a complex, powerful language for describing the interactions among web services. BPEL includes control structures that enable conditional processing and sophisticated error-handling routines, and its wide adoption makes it portable across a variety of platforms. Using BPEL, a developer can define which services perform particular tasks and how messages are exchanged among those services with as much control as if the workflow were hard-coded into the application.

The work by Kloppmann et al described in (35) and by Clement et al in (36) describes an extension to the BPEL specification called BPEL for People, which extends the BPEL language to include activities performed by people. BPEL for People was developed in conjunction with the Web Service-Human Task (WS-HumanTask) specification (37) enabling the composition of sophisticated workflows that include both web services and human services that can be described by WSDL.

2.2.3. WSMO

A WSMO service description includes a process model for the service it describes (which may include a complete business process), but like the other semantic service description specifications discussed above, WSMO does not include any specific mechanism for encoding an end-to-end process that incorporates different services, but its ability to represent a complete business process within a service description merits mention here.

2.2.4. METEOR-S

One project that was specifically designed to enable the type of workflow composition described here is the Managing End-to-End Operations – Semantic Web

Services (METEOR-S) project (13,14), which was specifically designed to enable the automated dynamic matching of services to complete a defined business process. METEOR-S uses a UML Activity Diagram to model interactions among services, although the types of semantic annotation of models tasks discussed here is not part of that research.

The work discussed in (38), part of the METEOR-S project, describes a hybrid approach where the developer begins with a defined process and a series of services that can be composed to complete the workflow. The process of adding services to the existing workflow is streamlined by applying the matchmaking and reasoning capabilities provided by the METEOR-S framework. This eases the burden on the developer, eliminating the need to manually comb through the available services to find those that may fit the newly identified need. One item of note within the workflow composition process as described is that specific human actions are called out as an essential part of the workflow execution, though there is no discussion of a formal description format for these human-based processes. While it is specifically mentioned, this important human interaction is glossed over because it is not a significant part of the METEOR-S project's focus. As will be discussed later, the participation of human actors within a workflow composition is a critical element that has not received sufficient attention.

2.2.5. OWLS-MX / SAWSDL-MX Families

OWLS-MX and SAWSDL-MX are two related research projects that have developed a series of hybrid service matchmakers based on the OWL-S and SAWSDL service specifications respectively. The OWLS-MX project (18) employs a hybrid

matchmaker that combines the semantic markup of OWL-S with a method for deducing semantic information from the syntactic similarity of terms in a service description. The OWLS-MX matchmaker takes five different measurements of semantic and syntactic similarity of service interfaces and generates a composite score to determine whether two service interfaces are matches for each other. During evaluation of the OWLS-MX matchmaker, the developers discovered that it had a tendency toward false positives, where services were incorrectly matched due to the syntactic similarity of terms that were not actually semantically compatible. The problem of false positives in OWS-MX was addressed in OWLS-MX2 (19) where a refinement of the techniques pioneered in the original OWLS-MX matchmaker resulted in a more reliable service matchmaking capability.

One limitation of the OWLS-MX matchmaker is that it was designed to perform matchmaking among services described using the OWL-S model; semantically annotated web services that employed a different model could not be used with the OWLS-MX family of matchmakers. This limitation was partially addressed by the development of SAWSDL-MX (16) and SAWSDL-MX2 (17), which applied the principles explored in the OWLS-MX family of matchmakers to the SAWSDL description specification. Results were comparable to those with the OWLS-MX series of matchmakers.

The concepts explored in OWLS-MX and OWLS-MX2 were enhanced and expanded with OWLS-MX3 (20,21), an adaptive variant of the OWLS-MX family of matchmakers. With OWLS-MX3, the developers added a machine learning element to the hybrid semantic matchmaker previously employed. In addition to employing semantic

and syntactic matching, OWLS-MX3 employs a binary classifier based on a Support Vector Machine (SVM). The SVM-based classifier is trained on an independent set of OWL-S service descriptions before being asked to find service matches. The aggregated results of the different OWLS-MX classifiers are averaged and the results of that are applied to the problem of classifying new services. While the developers judged that OWLS-MX3 did not perform significantly better than OWLS-MX2 in terms of either precision or recall, they felt that the ability to train the OWS-MX3 on a set of service descriptions entirely independent of the services it would later be called on to classify gave OWLS-MX3 a significant advantage over its forebears.

2.3. Optimization

While optimization has been a robust area of research for many years, very little work has been done specifically on the optimization of service compositions.

2.3.1. One-Dimensional

A simple, one-dimensional evaluation of alternatives is the easiest approach in that it makes it easy to compute the relative rankings of each composition and evaluate them against each other. If we consider a set of services that may be composed in different combinations, then optimizing on one dimension of QoS is a simple matter of calculating the QoS of each possible service combination and selecting the one that best meets the user's preferences.

The one-dimensional analysis problem may be complicated somewhat if the service providers are willing to negotiate the terms of their service, perhaps to offer a discounted cost for large-volume users. In this case, the service evaluation process is an

iterative one such as that described by McDowall and Kerschberg in (39). While such iteration makes the actual selection process more involved, calculating the optimal service composition remains a straightforward problem.

Cost

As an example, consider cost as the criteria for a one-dimensional analysis. The cost to invoke each service can be used to calculate the total cost of each service composition, resulting in a total cost for each candidate composition. If the user's only concern is minimizing cost, then the optimal selection is the composition with the lowest total cost.

2.3.2. Multi-Dimensional

In practice, most users would weigh several criteria when deciding which service composition best meets their needs. For example, selecting the composition with the best balance of cost and responsiveness may be important to the user. Alternatively, cost may not be a major concern but a combination of high security and fast response may be the primary consideration. Regardless of the specific criteria being evaluated, the evaluation is a multi-dimensional analysis problem and so is a significantly more complex problem than one-dimensional analysis.

Multi-dimensional analysis has been the subject of a great deal of research over the years, and the advent of digital computer systems makes it relatively easy to perform the sophisticated calculations necessary for multi-dimensional problems such as the service composition problem; a thorough overview can be found in Alodhaibi (40). The relative ease in applying multi-dimensional analysis to a broad array of problems has led

to a growth in the development of decision guidance systems that are designed to help users weigh the many factors that go into a complex decision, and quickly discover the best options available. The degree to which such a system calculates the optimal decision as opposed to approximating the optimal selection is generally a function of how long the calculations are allowed to continue. For complex problems, the computing time required to find the single best solution may outweigh the value to the user of narrowing the set of potential selections to some limited number of satisfactory solutions.

Narrowing the range of possible choices based on a set of evaluation criteria is common selection problem known as "top-k" selection, where k is the number of alternatives returned to the user. By choosing from among the k best solutions to the problem, the user is assured that any selection is among the best available solutions, but the user avoids the cost of an exhaustive evaluation. Because top-k selection is a statistical analysis problem, the analysis includes an associated probability that expresses how likely it is that all of the k selections are indeed within the overall top k. In general, as the required confidence increases the computational cost of the analysis increases. Minimizing this cost, while increasing the probability of correct selection, is the focus of ongoing research in the field of decision guidance systems.

One example of a multi-dimensional decision guidance application is the Composite Alternative Recommendation Development (CARD) framework described in Brodsky et al (41). The CARD framework was designed to recommend packages of services for a user, such as a combination of flight reservation, hotel, and rental car. While this is not explicitly recommending a combination of services to complete some

defined workflow, the principles applied are the same: there are several potential combinations of services that can be bundled to accomplish a task, and the user would like a recommendation as to which combination best meets the user's priorities. CARD was developed to work with both atomic services and composite services, which is a use comparable to a business process description.

The CARD system uses a knowledgebase to store information about service offerings and user preferences, and employs an extension of the Structured Query Language (SQL) called Decision Guidance SQL (DG-SQL) described in Brodsky and Wang (42). The CARD system uses DG-SQL to query the knowledgebase for service recommendations based on user preferences before the service selection process begins. User preferences are captured in a profile, and the system employs machine-learning techniques to refine its service selection process as users accept or reject different recommendations presented by the CARD system. One limitation of the CARD system is that service information is stored in the knowledgebase and must be periodically refreshed, limiting the currency of the data and limiting the evaluation criteria to those that are supported by the schema of the knowledgebase and the preferences the user expressed before the analysis process began.

Building on the CARD work, the Cluster-Optimizing-Diversity (COD) framework by Alodhaibi et al (43) extends the CARD recommender by using different utility functions to analyze service packages based on differing immediate needs of a user at any given moment. For example, a user may be examining two travel packages for different purposes: one for a business trip and one for a personal vacation. The criteria

used to evaluate each package may differ because of the different priorities for each purpose. When evaluating a business travel package, assuring the traveler arrives in time for a critical meeting may be more important than the price of the trip. Conversely, when evaluating a vacation package minimal price may be more important than arriving at the destination before a particular time.

Another innovation of the COD framework over its predecessor is that rather than soliciting user preferences before the analysis begins, COD learns the user's preferences based on feedback the user provides on each recommendation the COD system offers. COD also performs a more complex analysis process, evaluating any number of aspects of a service using an n-dimensional utility space, where different axes of value are used to evaluate which combination of services best meets the user's needs based in part on feedback the user has provided to previous recommendations. The COD framework recommends clusters of services based on utility functions that are solicited from the user during the analysis, reducing or eliminating the need to solicit user preferences before analysis begins, and adapting more quickly to changing user priorities. Like the CARD framework, the COD framework is limited by the service evaluation criteria that are stored within the knowledgebase before the analysis begins, so the analysis criteria are necessarily limited to those factors. Like the CARD framework, COD was not designed with process-based workflow compositions in mind but employs the same analysis principles and can be readily applied to the workflow analysis problem.

A contrasting approach is described by McDowall and Kerschberg (44) and (45), where social networks and service registries are used as the basis for developing a

recommendation. Instead of evaluating service offerings based on the representation of the service provider, social networks are used as the basis for forming a recommendation by querying the networks for other users' evaluations of the services and/or the service providers. This approach provides a near-real-time assessment of the public's satisfaction with a given service provider. This information can be analyzed on its own or incorporated with factors such as cost to develop a broader multi-dimensional profile of a given service that can be used as the basis for such an analysis.

As discussed in (44), depending on the source of the assessment information, negative information about service providers may be available to factor into the analysis. Negative information about their performance is not normally offered up by service providers, and so may be difficult to include in systems that base their assessment criteria on information available from service providers. One notable limitation of this work is that a lack of unambiguous links from service providers to their profiles in social networks or business registries often requires manual mapping between the service descriptions and the location where the assessment information is being queried.

3. OVERVIEW OF THE DRUID SERVICE COMPOSITION METHODOLOGY

This research addresses the limitations of previous work by defining Druid, a semantic service composition methodology that is suitable for composing both physical and digital services into an executable workflow based on a business process model, and recommending the optimal service composition based on QoS characteristics of each of the services, as well as the QoS of the aggregate recommended workflow. The Druid methodology includes a service description language suitable for describing both the syntax and semantics of the interfaces to either physical or digital services, including both SOAP and REST web services. The methodology also includes extensions to the BPMN modeling language necessary to define the semantics of a business process to enable automatically matching services to process activities. These BPMN extensions include a means to specify the task type of each activity comprising the process and, the input and output parameters from each activity. The methodology also includes a model for specifying QoS parameters of services and processes. Finally, the methodology includes a formal service composition optimization framework implemented using a mathematical programming model. When a business process model, semantic services, and QoS parameters are encoded in the Druid model, that information is passed to a process that computes the optimal service composition based on the QoS aspects of each service in each composition.

The subsections that follow provide an overview of the Druid methodology and describe at a high level how its parts work together. Formal definitions of these concepts, together with detailed explanations of the research contributions embodied in this methodology, are provided in subsequent chapters.

3.1. Overview of the Service Composition Methodology

To better understand the discussion that follows, it is helpful to define some foundational terms that will be used in the explanation. As this service composition process begins with process models defined using BPMN, that specification's terminology is used where appropriate.

A service is a means of completing some unit of work. Within the context of Service Oriented Architecture, "service" usually refers to an implementation-independent interface to software, but this research takes a broader view of a service as any means for completing work, whether the service is provided by software or by some physical means such as a person (for example, a plumber provides services such as repairing a leak).

Services can be subdivided into two main categories: atomic services and virtual services, defined below.

Atomic Services: There is no commonly accepted formal definition for an atomic service; for the purposes of this discussion an atomic service is the lowest level to which services are decomposed and is the level at which QoS metrics are assigned to services[4].

Virtual Services: In some cases, multiple atomic services may be composed together and offered through a single interface. This arrangement is known as a "virtual

[4] As a practical matter, the point at which a "service" can be defined has been moving lower in the 7-layer ISO stack, to the point where we are now speaking of "Infrastructure as a Service." This definition is therefore necessarily arbitrary for the purposes of this discussion.

service." The simplest virtual service is composed of individual atomic services, but it is also possible to compose a virtual service from combinations of other virtual services and atomic services. Given this definition, every business process also constitutes a virtual service and could be offered as such.

A BPMN model is a specification of a business process. A process is composed of individual steps, each of which is called an "activity." Each activity includes a semantic description, called a "task type," that categorizes the function or purpose of the activity (e.g., to reserve a hotel room). Task types are defined in an external ontology that is referenced from the process model. In Chapter 4, it is demonstrated that a business process model is a specification for a virtual service.

A summary of the Druid service composition methodology presented in this research is depicted in Figure 1. Each stage of the process is numbered to indicate its relative order, and each is described below.

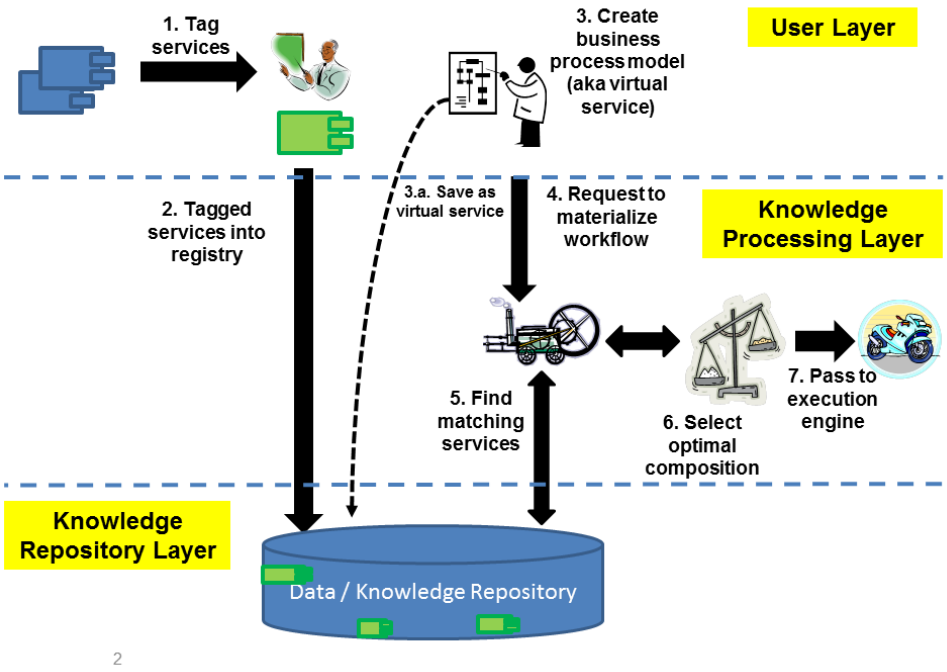

Figure 1: Druid Methodology Overview

Step 1 is to tag available services with the semantic metadata necessary to enable a matchmaker to match services to activities in a business process model. Matching services to activities entails verifying that the service performs the same task type that the activity represents, such as returning a weather forecast. Matching services to each other entails determining whether the outputs of any given service comprise all the required inputs of another given service, as this is necessary in order to compose the two services as part of a workflow. This is enabled by creating OWL-based service descriptions that conform to the Ogma service description language described in Chapter 7. These descriptions include semantic annotations for each service and for each input and output parameter of every service. In addition to the semantic information, each service

description includes other information necessary to use a service, such as binding information, supported communications protocols, and information about the service provider. Each service description is supplemented with QoS information encoded using the Ecne QoS model described in detail in Chapter 5.

In Step 2 of the methodology, the service descriptions are uploaded to a service registry to enable search and retrieval. This is the means to identify candidate services for the service composition.

Step 3 depicts a business analyst creating a semantically-annotated BPMN model that describes the business process to be automated. The semantic annotations appended to this business process model are encoded using the BPMN extensions described in Chapter 6, and includes semantic annotations indicating the task type of each activity in the BPMN process model, as well as annotations describing the input and output parameters of each activity in the model. It contains sufficient information to create a virtual service instance as defined formally in Chapter 4. Optionally, the business analyst may save the process specification as a virtual service for later use.

In Step 4, the annotated process model is submitted to a matchmaking engine that parses the BPMN model and extracts the semantic information from each of the activities in the model, as well as the ordering of the individual activities within the process model.

In Step 5, the matchmaking engine compares the semantic information about the activities in the process model to the service descriptions in the registry to find those services that can perform each of the activities specified in the process model. Once candidate services are found in the registry, their input and output parameters are

compared to ensure that for each service matched to a given activity, there are one or more corresponding services mapped to the immediately preceding and succeeding activities whose inputs and outputs are compatible.

After this initial filtering step, the QoS parameters for the services in each composition are retrieved, and the candidate service compositions are passed to the optimization processor in Step 6. The QoS information for each service is encoded in the Sucellos optimization model described in Chapter 5. The candidate service compositions are compared and a recommended optimal service composition is selected based on the QoS parameters.

Step 7 is the final step in the process, in which the selected optimal composition is passed to an execution engine. In the case of a composition consisting solely of web services, this execution step can be accomplished by encoding the composition using BPEL and passing it to a BPEL execution engine. In the case of a composition that includes both physical and web services, execution would require a more complex process where the web services are executed by a computer and the appropriate interactions with physical systems, including humans, are executed using more specialized computer-to-physical interfaces.

3.2. Service Composition by Example

The Druid methodology described above has several elements that must work together; the functionality of each of those elements must be understood in relation to the other elements that it supports. A brief description of each of these elements is provided

in the following sections, with formal definitions provided in Chapter 4 and detailed technical explanations in succeeding chapters.

All services, whether atomic or virtual, are semantically described by task types in the same manner that activities are described. These task types may be from the same ontology as the activity task types or from a separate ontology. Service descriptions also include semantic descriptions of each input and output parameter.

Service composition begins with a set of available services, both atomic and virtual. A developer or ontologist creates interface descriptions for each service, providing the semantic markup that is necessary for the matchmaking process. This corresponds to Step 1 of the Druid methodology (see Figure 1). These service compositions are uploaded to the service registry depicted in Step 2. For this weather example, assume that among these services are those shown in Table 1, which lists the service name, task type, and the semantic types of the input and output parameters.

Table 1: Example Services

Service	Task Type	Inputs	Outputs
convertLocation	locConvert	latitude, longitude	postalCode
transformLocation	locConvert	latitude, longitude	postalCode
getWeather	wxForecast	postalCode	forecast
getForecast	wxForecast	postalCode	forecast
returnWx	returnWx	postalCode	weatherData
changeFormat	changeFormat	weatherData	forecast

In Step 3, a business analyst specifies a process using BPMN. This example will use the process depicted in Figure 2, which shows a simple weather forecast process. This

process is composed of two activities: Convert Location and Get Weather. The order of these activities is specified by the arrows in the process model, which BPMN refers to as "sequence flows."

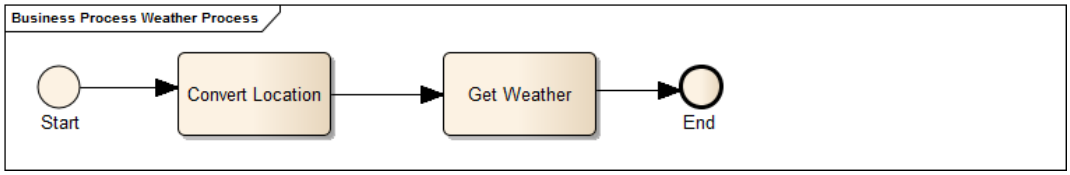

Figure 2: Sample Process

For each of the activities in this process, the business analyst assigns a task type that provides a reference to the type of work that activity represents. For this example, the Convert Location activity has a task type of "locConvert." Each activity also has a set of input parameters and a set of output parameters; each of these parameters is identified by the semantic type of the parameter. The set of inputs for the Convert Location activity is shown in Table 2 and the set of outputs for the Convert Location activity is shown in Table 3. The other activities have similar definitions.

Table 2: Convert Location Inputs

Parameter Name	Data Type	Semantic Type
lat	xs:String	latitude
lon	xs:String	longitude

Table 3: Convert Location Outputs

Parameter Name	Data Type	Semantic Type
zipCode	xs:String	myOntology#postalCode

At this point, the analyst could optionally upload a description of the business process specification to the registry as a virtual service. Storing the virtual service definition in the registry makes it available for use as a template (for others who may wish to implement the same process or something similar). It also makes the virtual service available for use in other service compositions.

In order to transform the BPMN process model into an optimized and executable workflow, each BPMN activity must be associated with one or more services. To accomplish this transformation, the BPMN specification is submitted to a service matchmaking engine in Step 4 of Figure 1; this engine performs the activity-to-service mapping. As formally defined in Chapter 4, an activity-to-service mapping is only valid if each of the services has the same task type as the activity and all of the inputs and outputs specified for the activity. Candidate services are selected based on an analysis of the service descriptions published to the service registry. This analysis is based on a comparison of the task type of each activity compared to the task type assigned to each service, as well as a comparison of each service's inputs to the parameters available from services matched to activities that occur earlier in the process specification.

Some of the selected services may be virtual services. In addition to the inputs, outputs, and task type that the virtual service performs, a virtual service encapsulates the set of activities that comprise the virtual service. Given that a virtual service is a process

model composed of activities, each of its activities can be mapped to services that are themselves virtual services. This mapping can be recursive, with any given service potentially mapped to a combination of atomic and virtual services. Eventually, each of the virtual services' activities is eventually decomposed down to individual atomic services. This decomposition results in a tree structure where each leaf of the tree is an atomic service and all other nodes in the tree are virtual services. An illustration of such a decomposition, using the weather example process and services, is depicted in Figure 3.

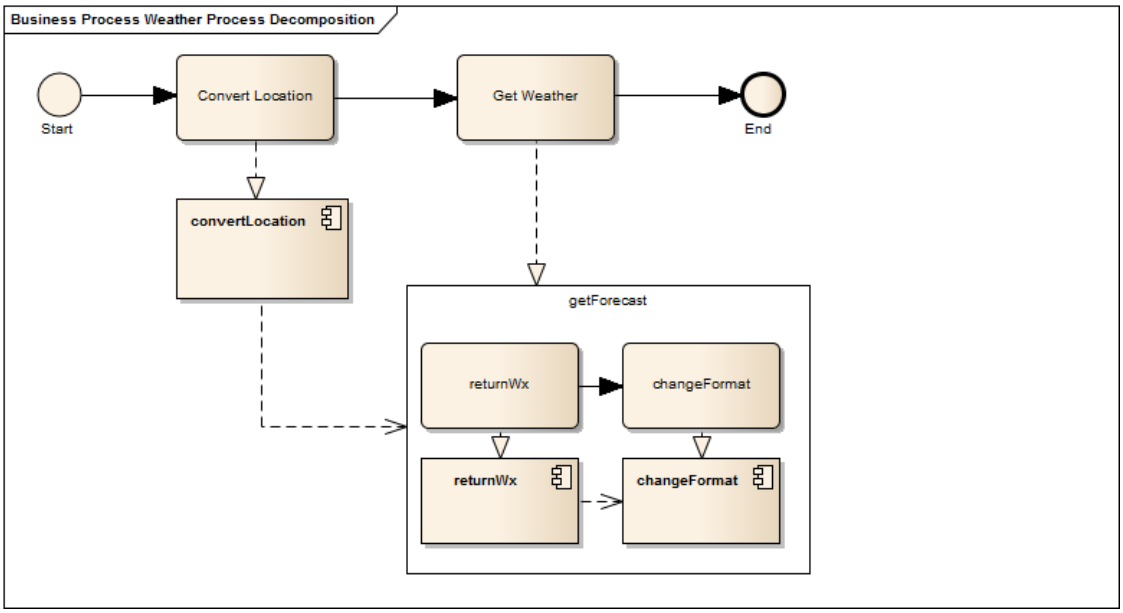

Figure 3: Weather Process Decomposition

Once all available services have been assessed and mapped to activities in the process model, and all virtual services are decomposed into their atomic services, we can determine whether any combination of atomic services can be composed into an

executable process that comprises all of the activities in the original business process model.

In order to compose services into an executable process, it is necessary that the inputs for each service in the composition be provided by a preceding service in the composition. This is determined by matching the semantic types of each input of each service to the semantic types of the outputs of preceding services. These outputs and inputs need not have the same name, but they must have the same semantic type. For example, if a location service has an output called “zip code” and a weather service has a single input called “postal code,” we can see intuitively that these two parameters have the same meaning. But in order for the matchmaker to match them, the parameters must have the same semantic type. An individual service input or output parameter, together with associated semantic and type metadata, is called a “semantic parameter.” For example, a data element called “zipCode” would include a semantic annotation that references an ontology and a type annotation indicating it is stored as a string.

Once each activity is associated with one or more services, and the services’ inputs and outputs have been compared semantically, we can compute a set of candidate service compositions. A candidate service composition consists of a set of atomic services that can be composed to materialize the process originally specified in the BPMN process model. Calculating the set of candidate service compositions is a straightforward matter if the services are represented as a directed graph. First, consider each service that has been mapped to an activity as a node in the graph. For each case in which one service’s outputs provide the inputs required by another service, assert an edge

from the former service to the latter service. The result is a directed graph of the services, where each path from the first activity to the final activity constitutes a candidate service composition.

An example of such a graph is shown in Figure 4 (for simplicity, this example shows only atomic services).

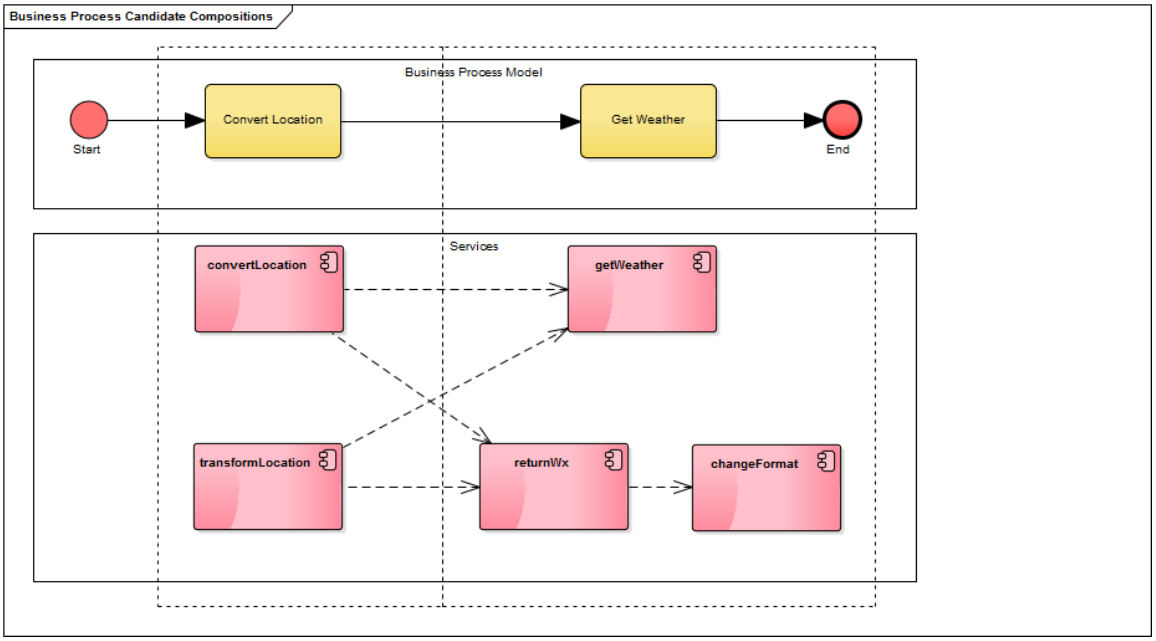

Figure 4: Candidate Service Compositions

The candidate compositions that can be assembled to complete the process are discovered by finding all the paths from services matching the first activity to those matching the final activity. In the example depicted in Figure 4, there are four candidate service compositions:

- convertLocation-getWeather

- convertLocation-returnWx-changeFormat
- transformLocation-getWeather
- transformLocation-returnWx-changeFormat

Finding these paths through the directed graph of services can be accomplished using common graph analysis algorithms such as Floyd-Warshall (46). This matchmaking and path analysis corresponds to Step 5 of the DRUID methodology.

Once all of the candidate service compositions have been enumerated, it is possible to calculate the QoS of each composition. The QoS parameters, of each service in each of these candidate compositions, are retrieved from the service description, and the QoS parameters for each candidate composition are passed to the QoS optimization processor. The QoS optimization processor compares each of the candidate service compositions and recommends the optimal composition of each service and the overall workflow, as determined by the user's preferences (e.g., minimize cost). The QoS analysis corresponds to Step 6 of the DRUID methodology.

Upon completion of the QoS analysis, the optimization processor returns a recommended optimal service composition based on the QoS parameters. This recommended composition may then be passed to an execution engine. This is Step 7 in the DRUID methodology.

3.3.Service Composition System Architecture

In order for the Druid Methodology to be effective, a number of system components must work together.

A notional architecture of such a system is depicted in Figure 5 as a UML model. The following subsections describe each of the components shown, and how they work together.

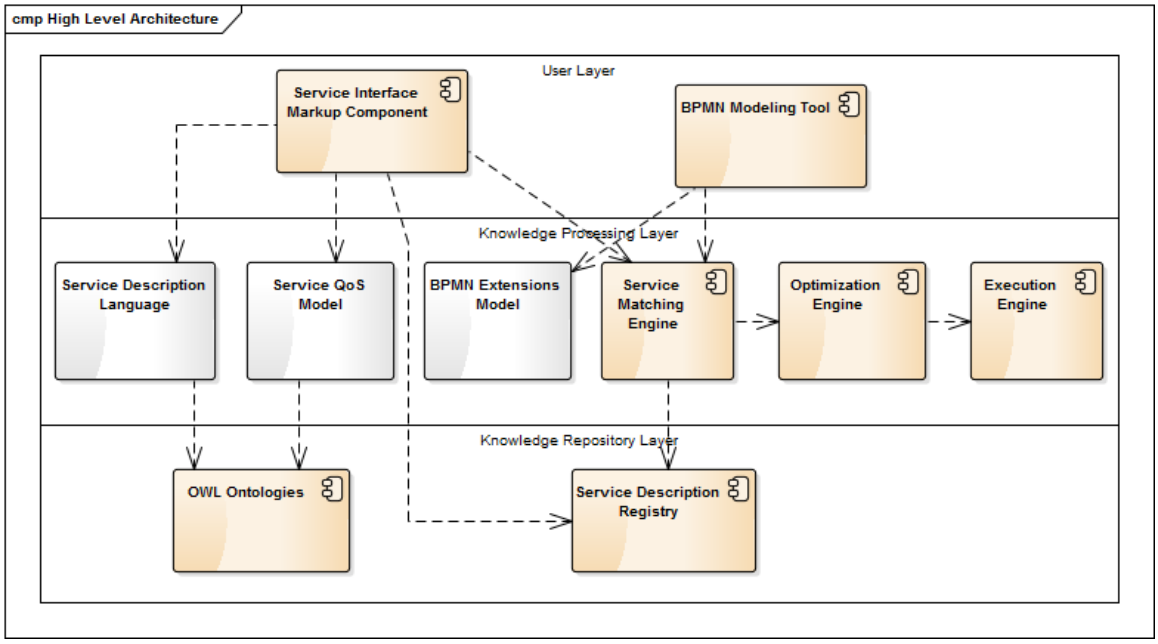

Figure 5: Service Composition Architecture

3.3.1. User Layer

The User Layer provides user-facing components. In Figure 5, we note that this layer includes two major components, an interface for creating the service interface markup and a BPMN modeling tool.

The Service Interface Markup Component allows the user to semantically describe a service interface using the service description language – OGMA – described

in detail in Chapter 7. The user employs this tool to create an OWL-based service interface description and then submits it to the Service Description Registry.

The BPMN Modeling Tool allows a business analyst to specify a business process. This process specification includes annotations to the BPMN model that conform to the BPMN extensions described in Chapter 6. This process specification will be passed to the Matchmaking Engine in the Knowledge Layer.

3.3.2. Knowledge Processing Layer

The Knowledge Layer in Figure 5 encompasses the functions necessary to transform a BPMN-based activity workflow specification into a collection of services to support the execution of an optimal workflow. The Service Description Language component – specified and developed as part of this research – defines the metamodel underlying the service description language of Chapter 7.

Another component within the Knowledge Layer is the Service QoS Model. This model is an OWL-based specification and is described in detail in Chapter 5. This model is distinct from, but readily integrated with, the Service Description Model in order to describe the QoS metrics of a service as part of the service interface description.

The BPMN Extensions Model within the Knowledge Layer defines the extensions to BPMN that were developed as part of this research. These model extensions are necessary to allow the addition of semantic information necessary to match service descriptions to process activities and are described in Chapter 6.

The Service Matching Engine and Optimization Engine are the main processing components of the Knowledge Layer. The Matchmaking Engine parses both the BPMN

process specification and the service interface descriptions, and processes them to complete the activity-to-service mapping which is the first step in developing an optimal service composition. The Matchmaking Engine then computes the possible service compositions by comparing the inputs and outputs of adjacent services as described above. After the possible service compositions have been computed, they are passed to the Optimization Engine where the optimal service composition is selected by computing the aggregate QoS metrics of each candidate composition and comparing them to determine which composition is optimal based on the user's preference (e.g., to minimize cost). A detailed explanation of how QoS metrics are aggregated is provided in Chapter 4.

The final component of the Knowledge Layer is the Execution Engine. Once the Optimization Engine has selected an optimal service composition, this composition can be mapped to an execution language such as BPEL, and passed to the Execution Engine.

3.3.3. Knowledge Repository Layer

The Data Layer includes the Service Description Registry and the Service Description and QoS OWL Ontologies. The Service Description Registry is a data store that holds descriptions of every atomic and virtual service that can be used to develop a service composition. Because the service descriptions are OWL-based, the Service Description Registry may be instantiated using any appropriate database, for example in the MarkLogic XML database or the Sesame RDF database.

The OWL Ontologies are the service description language definition and the QoS model definition, together with any supplemental ontologies used to define task types or

semantic parameters. These ontologies provide the semantic foundations that enable the processing performed by the Matchmaking Engine and the Optimization Engine.

3.3.4. User View

The users' view of the Druid service composition methodology is shown in Figure 6, which depicts a developer and business analyst each using different elements of the methodology.

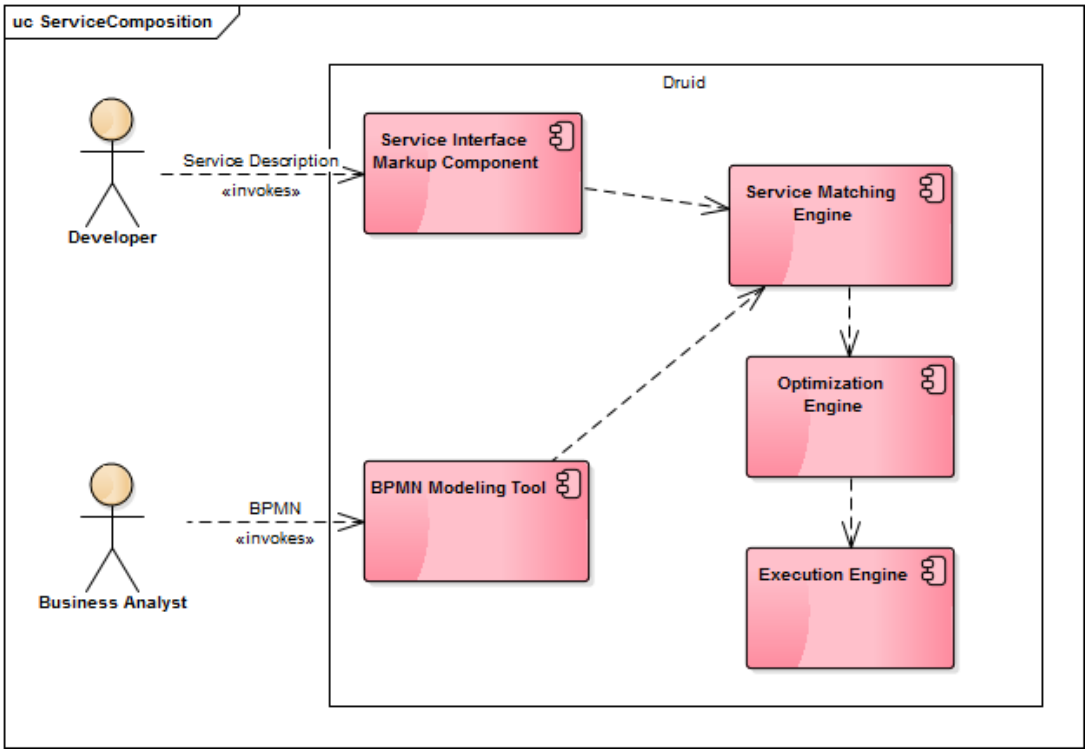

Figure 6: Service Composition Methodology Usage

Figure 6 depicts a Developer creating service interface descriptions and a Business Analyst independently developing a BPMN process specification. The service

descriptions and process specification are passed to the Matchmaking Engine, then the Optimization Engine, and ultimately to the Execution Engine.

4. FORMAL OPTIMIZATION SERVICE COMPOSITION FRAMEWORK

To compare multiple candidate service compositions and determine the optimal composition requires two things: a formal model of services and processes that allows a precise comparison, and a QoS model that describes the parameters to be used in the comparison and how they will be aggregated. The description that follows mirrors the description provided by McDowall, Brodsky, and Kerschberg in (47).

4.1. Terminology

Chapter 3 presented an intuitive description of process models and service composition; this chapter presents the formal definitions that are the foundation for the methodology. The important intuitive definitions are summarized in Table 4 below. The formal definitions are provided in this chapter.

Table 4: QoS Term Definitions

Term	Definition
Process	A specification of some business function a user desires to complete
Activity	A discrete element of a process that may be reused across multiple processes
Task Type	The semantic description of an activity or service that provides a reference to the type of work performed by an activity or a service
Virtual Service	A service offering that provides a single interface to what may be a more complex service composition or orchestration
Atomic Service	The lowest level of service decomposition

In order to assess the QoS of a service composition, it is first necessary to formally define how individual services are composed into a business process that conforms to some business process specification expressed as a BPMN model. This specification must be sufficiently detailed to allow each activity in the BPMN model to be mapped to one or more atomic services.

4.2. Optimal Service Composition

Intuitively, the service composition optimization problem is as follows: given a desired process, a set of services, constraints, and an objective such as minimizing cost, select the set of services that comprise the process and best meets the objective within the constraints.

The formal definition of service composition based on a process model is as follows (this discussion mirrors the intuitive discussion in Chapter 3).

Let $\mathbb{T} = \{t_1, \dots, t_n\}$ denote a set of task types. For example, $t_i (1 \leq i \leq n)$ can be the task type "reserve a hotel room." If a service performs the same function as an activity in the process model, then we say they have the same task type.

Let SP denote the set of all semantic parameters.

Definition 1: Definition of a Service

Definition: Given $\mathbb{T}$ and SP, a service $\mathcal{s}$ is a tuple $< id, I, O, T >$

Where

id is a unique identifier

$I \subseteq SP$ is the set of input semantic parameters

$O \subseteq SP$ is the set of output semantic parameters

$T \in \mathbb{T}$ is the task type that describes this service

The above definition includes a unique identifier so that similar services offered by different providers can be distinguished from each other. Such a tuple defines sufficiently an atomic service.

A virtual service, defined below, is needed to enable the recursive composition of services.

Definition 2: Definition of a Virtual Service

Definition: A virtual service $\mathcal{s}$ (also called a process) is a tuple

$$\mathcal{s} =< id, I, O, T, A, DG, aTask: A \rightarrow \mathbb{T}, S >$$

Where

id is a unique identifier

$I \subseteq SP$ a set of input semantic parameters

$O \subseteq SP$ a set of output semantic parameters

$T \in \mathbb{T}$ is a Process Task Type associated with the virtual service $\mathcal{s}$

$A = \{a_1, \ldots, a_n\}$ is a set of activities used in $\mathcal{s}$

$DG \subseteq A \times A$ is an activity precedence graph that must be acyclic. $(a_1, a_2) \in DG$ (also denoted $a_1 \prec a_2$) means that activity a_1 must precede activity a_2

aTask: A $\rightarrow \mathbb{T}$ is a mapping that associates every activity $a \in A$ to its task type $t = \text{aTask}(a)$ in $\mathbb{T}$

$S = \{s_1, \ldots, s_n\}$ is a set of services that can be used by activities in A

Note that a virtual service is a service, and any service in S may itself be a virtual service. Therefore, multiple services can be used for each activity in a virtual service. A particular instantiation is formalized in the following definition.

Definition 3: Definition of Service-to-Activity Mapping

Definition: Given a virtual service $\mathcal{s} =< id, I, O, T, A, DG, aTask: A \rightarrow \mathbb{T}, S >$, an activity-to-service mapping A2S: A $\rightarrow S$ is a mapping that associates each activity in A with a service in S, that must satisfy the following properties:

Let sTask(s) denote the task T associated with service s in S;

let SI(s) denote the input set I associated with s;

let SO(s) denote the output set O associated with s;

The A2S mapping must satisfy:

$$(\forall a \in A)\mathrm{aTask}(a) = \mathrm{sTask}(\mathrm{A2S}(a))$$

$$(\forall a \in A)\mathrm{SI}\big(\mathrm{A2S}(a)\big) \subseteq \mathrm{PrecOut}(a)$$

Where

PrecOut(a) denotes the outputs of the services preceding service a

$$\mathrm{PrecOut}(a) = \bigcup_{b<a} \mathrm{SO}(\mathrm{A2S}(b))$$

(i.e., b is the set of all outputs produced by activities / services that precede a)

The notion of a virtual service instance, defined below, describes a recursive mapping of activities to available services for a given virtual service $v\mathcal{s}$.

Definition 4: Definition of a Virtual Service Instance

Definition: Let AS be the set of atomic services and VS be the set of virtual services. A virtual service instance (VSI) over (AS,VS) is a tuple $V =< \mathbb{S}, v_{\mathcal{S}}, \{A2S_{\mathcal{S}}\}_{\mathcal{S}\in\mathbb{S}\cap VS} >$

Where:

$\mathbb{S} \subseteq AS \cup VS$

$v_{\mathcal{S}} \in \mathbb{S} \cap VS$

$\{A2S_{\mathcal{S}}\}_{\mathcal{S}\in\mathbb{S}\cap VS}$ is a set of activity-to-service mappings $A2S_{\mathcal{S}}: \mathcal{S}.A \rightarrow \mathcal{S}.S$ where $\mathcal{S}.A$ and $\mathcal{S}.S$ are the set of activities and services of $\mathcal{S}$, respectively

Such that the following conditions are satisfied:

1. $\mathcal{S}.S \in \mathbb{S}$
2. $(\forall \mathcal{S}.S)SI(\mathcal{S}) \subseteq PrecOut(\mathcal{S})$

   Where

   $PrecOut(\mathcal{S})$ denotes the outputs of the services preceding service $\mathcal{S}$

   $$PrecOut(a) = \bigcup_{b<a} SO(b)$$

   (i.e., b is the set of all outputs produced by services that precede a)
3. $\neg(\exists \mathcal{S}_1, \mathcal{S}_2 \in \mathbb{S})(\exists a_1 \in \mathcal{S}_1.A)(\exists a_2 \in \mathcal{S}_2.A)(A2S_{\mathcal{S}_1}(a_1) = A2S_{\mathcal{S}_2}(a_2))$

   (i.e., no two activities with services of $\mathbb{S} \cap VS$ can be mapped via A2S to the same virtual service

We would like to find the "optimal" virtual service instance from among those available. To do this, we establish several quality of service (QoS) factors that can be used to express the utility to be optimized. Formal definitions for each of these metrics are provided below. These QoS metrics are described further in the Sucellos QoS model discussed in Section 5.

Given an atomic service $\mathcal{s}$, the QoS metrics being considered are cost, duration, rating, and unity; these are denoted C($\mathcal{s}$), D($\mathcal{s}$), R($\mathcal{s}$), and unity($\mathcal{s}$) respectively. We consider the QoS metrics for atomic services are given, and we define QoS metrics for a virtual service. The definition of the cost of a virtual service instance is provided below.

Definition 5: Definition of Service Cost

Definition: Given a virtual service instance $V = < \mathbb{S}, v\mathcal{s}, \{\text{A2S}_{\mathcal{s}}\}_{\mathcal{s} \in \mathbb{S} \cap \text{VS}} >$ over (AS,VS), the cost of $\mathcal{s}, \forall \mathcal{s} \in \mathbb{S}$, denoted cost($\mathcal{s}$), is defined recursively as follows:

$(\forall \mathcal{s} \in \mathbb{S} \cap \text{AS})$ cost($\mathcal{s}$) = C($\mathcal{s}$) where C($\mathcal{s}$) is the cost of atomic service $\mathcal{s}$

$(\forall \mathcal{s} \in \mathbb{S} \cap \text{VS})\ \text{cost}(\mathcal{s}) = \sum_{\text{a} \in \mathcal{s}.\text{A}} \text{cost}\big(\text{A2S(a)}\big)$

The cost of V, denoted $\text{cost}(V)$, is defined as $\text{cost}(V) \stackrel{\text{def}}{=} \text{cost}(v\mathcal{s})$

The duration of a virtual service instance, which intuitively is the expected time for the entire composition to run from initiation until completion of all services within the virtual service instance, is defined next.

Definition 6: Definition of Service Duration

Definition: Given a virtual service instance $V =< \mathbb{S}, v\mathcal{s}, \{\text{A2S}_{\mathcal{s}}\}_{\mathcal{s}\in\mathbb{S}\cap\text{VS}} >$ over (AS,VS) the duration of $\mathcal{s}, \forall\mathcal{s} \in \mathbb{S}$, denoted duration($\mathcal{s}$), is defined recursively as follows:

($\forall\mathcal{s} \in \mathbb{S} \cap$ AS) duration($\mathcal{s}$) = D($\mathcal{s}$) where D($\mathcal{s}$) is the duration of atomic service $\mathcal{s}$

($\forall\mathcal{s} \in \mathbb{S} \cap$ VS) duration($\mathcal{s}$) = $\max\{endtime(a)|a \in A\}$

where endtime(a) is defined as follows:

If $a \in A$ does not have a preceding activity ($i.e., Prec(a) = \emptyset$):

$\text{endtime(a)} \stackrel{\text{def}}{=} \text{duration}(A2S_{\mathcal{s}}(a))$

Otherwise:

$\text{endtime(a)} \stackrel{\text{def}}{=} \max\{\text{endtime(b)} + \text{duration}(\text{A2S}_{\mathcal{s}}\text{(a)}) \text{ |b} \in \text{ Prec(a)}\}$

The duration of V, denoted as duration(V), is defined as duration(V) $\stackrel{\text{def}}{=}$ duration($v\mathcal{s}$)

The rating of a service is a measure of users' ratings of a service, such as rating a service on a scale of 1 to 10. We assume that each service's individual rating has been normalized to the range $\{0..1\}$. The notion of rating for a virtual service instance is defined below.

Definition 7: Definition of Service Rating

Definition: Given a virtual service instance $V = < \mathbb{S}, v\mathcal{s}, \{\text{A2S}_{\mathcal{s}}\}_{\mathcal{s} \in \mathbb{S} \cap \text{VS}} >$ over (AS,VS) and the rating of each atomic service, the rating of $\mathcal{s}$ is denoted $r(\mathcal{s})$ and is defined as follows:

$(\forall \mathcal{s} \in \mathbb{S} \cap \text{AS})$ $r(\mathcal{s}) = \text{R}(\mathcal{s})$ where $\text{R}(\mathcal{s})$ is the rating of atomic service $\mathcal{s}$

$(\forall \mathcal{s} \in \mathbb{S} \cap \text{VS})$ $R(\mathcal{s}) = \frac{\sum_{\text{a} \in \mathcal{s}.\text{A}} sat(\text{A2S(a)})}{|\mathcal{s}.A|}$

Where

$|\mathcal{s}.A|$ is the number of activities

The rating of V, denoted $r(V)$, is defined as $sr(V) \stackrel{\text{def}}{=} r(v\mathcal{s})$.

The rating of any service or collection of services is therefore a value in the range of {0..1}.Knowing how to calculate each of the QoS parameters across a service composition, we can define the optimal service selection.

Definition 8: Definition of an Optimal Service Composition

Definition: Given the following input:

- Sets AS and VS of atomic and virtual services respectively
- A root service $rs \in VS$
- An objective expressed as a function

  $O: D(cost) \times D(duration) \times D(rating) \rightarrow \mathbb{R}$ that gives a value $O(C, D, R)$ for cost C, duration D, and rating R
- Minimum or maximum
- Constraint $\mathbb{C}$ is a Boolean expression in terms of C, D, and R that defines

  $\mathbb{C}: D(cost) \times D(duration) \times D(rating) \rightarrow \{true, false\}$

An optimal virtual service instance vsi is defined as

$vsi \stackrel{\text{def}}{=} aggmin_{i \in VSI}\ O(C(i), D(i), R(i))$ where VSI is the set of all virtual service instances over (AS, VS) with root service rs subject to $\mathbb{C}(C(i), D(i), R(i))$ where minimum is required. The definition is similar for the case where a maximum is required.

Example

To illustrate, consider the simple weather process presented in Section 3. In this example, it is necessary to complete two actions: convert the current location designation into a format accepted by the weather service, and retrieve the current weather for that location. This process is expressed in BPMN as depicted in Figure 7:

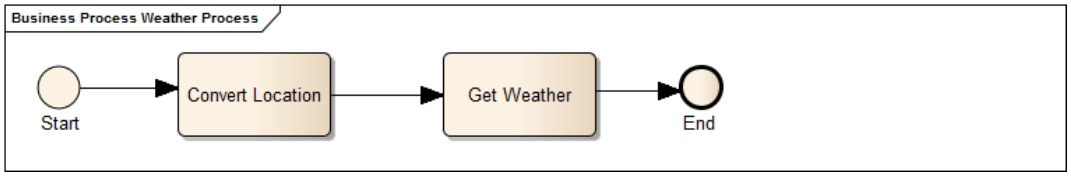

Figure 7: Sample Weather Process

In this example, the set of task types $\mathbb{T}$ is {locConvert, getWeatherReport } and the set of semantic parameters *SP* is {lat, lon, zipCode, currTemp, and currHumidity}. The set of input parameters *I* is {lat, lon, zipCode} and the set of output parameters *O* is { zipCode, currTemp, and currHumidity}.

Detailed information about the semantic parameters is summarized in the tables below. Table 5 defines the inputs *I* to the activity Convert Location:

Table 5: Convert Location Inputs

Parameter Name	Data Type	Semantic Type
lat	xs:String	myOntology#latitude
lon	xs:String	myOntology#longitude

Table 6 defines the outputs *O* of the activity Convert Location:

Table 6: Convert Location Outputs

Parameter Name	Data Type	Semantic Type
zipCode	xs:String	myOntology#postalCode

Table 7 defines the inputs *I* of the activity Get Weather:

Table 7: Get Weather Inputs

Parameter Name	Data Type	Semantic Type
zipCode	xs:String	myOntology#postalCode

Table 8 defines the outputs *O* of the activity Get Weather:

Table 8: Get Weather Outputs

Parameter Name	Data Type	Semantic Type
currTemp	xs:float	myOntology#temperature
currHumidity	xs:integer	myOntology#humidity

The set of activities *A* is {Convert Location, Get Weather}. The directed graph *DG* is $Convert\ Location \prec Get\ Weather$.

Each Activity in the model is mapped to a Task Type by the relation aTask. For example, the "Convert Location" activity has a Task Type of "locConvert":

$$aTask(Convert\ Location) = locConvert$$

The set of services *S* is {locationConverter, wxReporting}.

When combined, these elements fulfill the definition of a virtual service $\mathcal{s}$; this is summarized in Table 9.

Table 9: Virtual Service for Weather Reporting

Parameter	Value
id	2323452345
I	{lat, lon}
O	{currTemp, currHumidity}
T	getCurrentWeather
A	{Convert Location, Get Weather}
DG	$Convert\ Location \prec Get\ Weather$
aTask	Convert Location → locConvert; Get Weather → getWeatherReport
S	{locationConverter, wxReporting}

For the set of services *S* to be composed into a virtual service instance it must satisfy the conditions specified above (i.e., $(\forall a \in A)aTask(a) = sTask(S2A(a))$ and $(\forall a \in A)SI(S2A(a)) \subseteq PrecOut(a)$). To demonstrate this, the services locationConverter and wxReporting are summarized in Table 10.

Table 10: Service Summary

Service	sTask	SI	SO
locationConverter	locConvert	lat, lon	zipCode
wxReporting	getWeatherReport	zipCode	currTemp, currHumidity

As a comparison of Table 9 and Table 10 shows, the conditions for S2A mapping are satisfied and yield the mappings shown in Table 11.

Table 11: A2S Mapping for Weather Process

Service	Activity
locationConverter	Convert Location
wxReporting	Get Weather

The result is the VSI shown in Table 12:

Table 12: VSI for Weather Process

Parameter	Value
$\mathbb{S}$	locationConverter, wxReporting
$v_{\mathcal{S}}$	Weather Process
$\{S2A_s\}_{s \in \mathbb{S} \cap VS}$	locationConverter → Convert Location, wxReporting → Get Weather

The following sections describe the QoS model developed as part of this research and how that model is applied to determine the optimal service composition. These intuitive definitions are based on the formal definitions provided in Section 4.2.

4.3.Mathematical Programming Formulation

Using the above definitions, assessing the optimal service composition is a matter of calculating the aggregate QoS measures of each VSI and applying the definition of an optimal VSI described. To perform this, the definitions described above are instantiated using IBM's Optimization Programming Language (OPL) as described below.

The implementation begins by initializing the data required to perform the optimization calculations. The initialization code is shown in Table 13. The data initialization closely parallels the formal definitions in Section 4.2.

Table 13: OPL Data Initialization

```
{string} SP = ...;
{string} Aservices = ... ; /* ids, a subset of services */
{string} Vservices = ...;  /* ids, a subset of services */
float activationCost[Aservices] = ...;
{string} Services = Aservices union Vservices;
{string} Tasks = ...;
{string} Inputs[Services] = ...;
{string} Outputs[Services] = ...;
string task[Services] = ...;
{string} Activities[Vservices] = ...;
```

The data initialization first defines the set of semantic parameters as `SP`, the set of atomic services as `Aservices`, and the set of virtual services `Vservices`; each of these is defined as a set of strings. Next, the cost of each atomic service is defined as `activationCost`, an array of floats over `Aservices`.

The set `Services` is defined as the untion of the sets `Aservices` and `Vservices`. This is followed by defining the task types as the set `Tasks`. In the following two lines, the set of arrays `Inputs` defines the semantic parameters that are inputs to each service as an array of semantic parameters over the set of services with a similar definition for the set of arrays `Outputs`. Next, the array `task` associates task types with each service followed by the set of arrays `Activities` that defines the activities within each virtual service in `Vservices`.

After the initial data values are defined, the OPL model calculates additional data values that are necessary to perform the optimization computations. These code that performs these calculations is shown in Table 14.

Table 14: OPL Data Computation

```
tuple serviceActivityPair {
   string service;
   string activity;
};
{serviceActivityPair} VserviceActivityPairs =
   {<s,a> | s in Vservices, a in Activities[s] };

{string} PrecActivities[VserviceActivityPairs] = ...;
string aTask[VserviceActivityPairs] = ...;
string rootVservice = ...;

tuple serviceActivityService {
  string service;
  string activity;
  string mappedService;
}

{serviceActivityService} VserviceActivityServiceTuples =
   {<s,a,ms> | s in Vservices, a in Activities[s], ms in Services
                  : task[ms] == aTask[<s,a>] && s != ms };
```

This code section first defines a data structure `serviceActivityPair`, which is a set of tuples composed of one service and one activity. These tuples are used to define `VserviceActivityPairs`, which associate activities with virtual services in order to specify the activities within each virtual service.

Next, `PrecActivities` is defined as set of arrays over `VserviceActivityPairs`, this encodes the precedence graph DG defined in Section 4.2 by listing the activity preceding each activity within each virtual service.

After this, `aTask` defines an array of strings over `VserviceActivityPairs` that associates a task type with each activity in each virtual service. The string `rootVservice` specifies which of the `Vservices` is designated the root service.

The final two data structures defined are the tuple `serviceActivityService`, which captures a service, an associated activity, and a mappedService. This tuple is used to build the array `serviceActivityServiceTuples`, which maps a service to each activity in each virtual service, fulfilling the function of the *A2S* mapping defined above.

Having defined all of the data structures required in the optimality computation, it is appropriate to define the decision variables and decision expressions that will be used to compute the optimal service composition. These structures are defined in the code listing in Table 15.

Table 15: OPL Decision Expressions

```
dvar boolean s2a[VserviceActivityServiceTuples];

dexpr int noInvocations[s in Vservices] =
   sum (sas in VserviceActivityServiceTuples: sas.mappedService
   == s) s2a[sas];

dexpr int noInvPerVservice[v in Vservices][s in Aservices] =
   sum (sas in VserviceActivityServiceTuples: sas.mappedService
   == s && sas.service == v) s2a[sas];

dexpr float vServiceCost[v in Vservices] = sum (s in Aservices)
   activationCost[s] * noInvPerVservice[v][s];

dexpr float rootVserviceCost = vServiceCost[rootVservice] +
   sum(sas in
   VserviceActivityServiceTuples)vServiceCost[sas.mappedService];

dexpr float totalCost = vServiceCost[rootVservice];
```

The decision variable `s2a` is an array of Booleans over the set of `VserviceActivityServiceTuples`. The s2a variable is 1 if the tuple contains a virtual service, and activity, and a service mapped to that activity within that virtual service; it is 0 otherwise.

The decision expression (dexpr) `noInvocations` is an array of integers over the range of virtual services; it counts the number services that have been mapped to activities in a particular virtual service. The decision expression `noInvPerVservice` is a two-dimensional array over virtual services and atomic services; its purpose is to count the number of times any given atomic service is invoked within a given virtual service instance.

The decision expression `vServiceCost` is an array of floats over the set of virtual services that captures the cost of each virtual service instance for each virtual service in accordance with the definition detailed in Definition 5. By the same token, the decision expression `rootVserviceCost` is a float that captures the cost of the virtual service that has been designated the root virtual service. Finally, the decision expression `totalCost` is a float that captures the cost of the root virtual service.

The code listing in Table 16 shows the application of the decision expressions and constraints to calculate the optimal service composition.

Table 16: OPL Optimization Calculation and Constraints

```
minimize rootVserviceCost;

constraints {

forall (v in Vservices, a in Activities[v])
   sum ( ms in Services : task[ms] == aTask[<v,a>] && v != ms)
   s2a[<v,a,ms>] <= 1;

forall (v in Vservices, a in Activities[v]) (noInvocations[v] ==
   1) =>
      sum ( ms in Services : task[ms] == aTask[<v,a>] && v != ms)
   s2a[<v,a,ms>] == 1;

forall (v in Vservices) 0 <= noInvocations[v] <= 1;

forall (a in Activities[rootVservice])
   sum ( ms in Services : task[ms] == aTask[<rootVservice,a>] &&
   rootVservice != ms) s2a[<rootVservice,a,ms>] == 1;

forall (v in Vservices, a in Activities[v], ms in Services, i in
   (Inputs[ms] diff Inputs[v]) : task[ms] == aTask[<v,a>] && v !=
   ms) {
  s2a[<v,a,ms>] <=
   sum (precA in PrecActivities[<v,a>], precMs in Services, o in
   Outputs[precMs] : aTask[<v,precA>] == task[precMs] && o == i)
      s2a[<v,precA,precMs>];
}

rootVserviceCost >= 0;
```

In this example, the utility function described in Definition 8 is expressed as minimize `rootVserviceCost` based on the computation of that cost in the code listed in Table 15. This optimization is subject to the constraints shown above. The first constraint requires that every activity in each virtual service have at least one service mapped to that activity and that the service and activity have the same task type. The next constraint ensures that for each activity in a given virtual service, a service mapped to that activity have the same task type as the activity and that the activity is not mapped to

the virtual service that contains the activity (i.e., it ensures the mapping of activities to services is acyclic).

The third constraint ensures that any given virtual service is mapped to an activity only once. This is followed by a constraint that ensures all activities in the rootVservice have been mapped to services. The final constraint ensures that for each service mapped to an activity, the inputs of that service are provided by the outputs of a service mapped to a preceding activity, as required by the definitions shown in Definition 3 and Definition 4. Experimental results of this implementation are described in Section 8.2.4.

The OPL implementation and optimization takes place in Step 6 of the process, as highlighted in Figure 8.

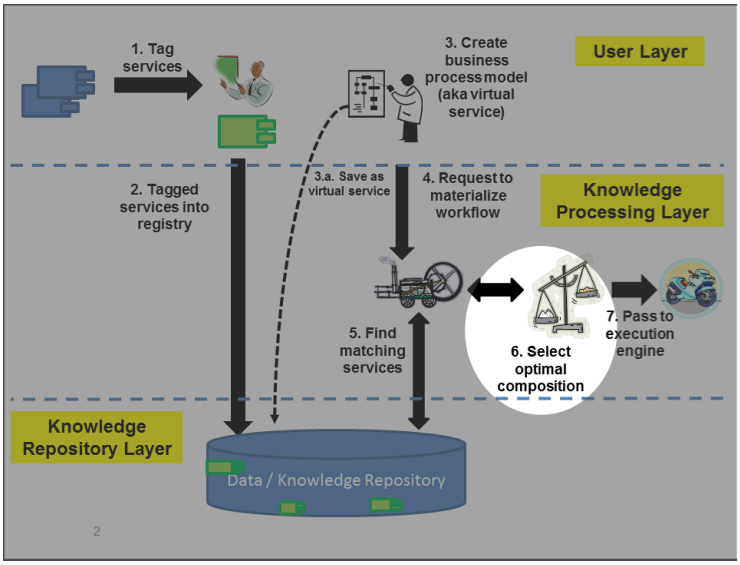

Figure 8: Composition optimization

5. SUCELLOS: A QUALITY OF SERVICE MODEL

This research employs a formal QoS model, called SUCELLOS, to optimize service compositions based on the QoS attributes of the individual services in a VSI. The QoS model is based on work done at France's INRIA research institute (26,48,49). The INRIA model is composed of four layers:

- A *QoS Core* ontology that describes the foundational concepts used within the remainder of the QoS model.
- An *Infrastructure QoS* ontology that describes infrastructure-specific QoS aspects such as processing power in the service hosting environment.
- A *Service QoS* ontology that describes QoS aspects of a particular service implementation.
- A *User QoS* ontology that describes the environment the user will be using to invoke services (e.g., using a smartphone vs. a desktop).

The original INRIA work was focused on measuring QoS as delivered to the user and then offering options to optimize QoS based (in part) on the user's environment. That is, the INRIA model assumes that all services are implemented as web services. In contrast, this research employs a model that operates at a more abstract level, where the implementation details of the service are much less important than a measure of the service's desirability based on factors such as price, responsiveness, community rating,

and similar measures. In short, SUCELLOS is a model that describes QoS for an abstract service. The SUCELLOS model is equally suitable for both IT and physical non-web services. In short, a QoS model describes not only the QoS for an abstract service that may be implemented in several ways, but also the QoS of non-Web services, such as physical services.

5.1.Description of the SUCELLOS QoS Model

The QoS metrics are associated with service descriptions that are defined during the process of service tagging. This step of the methodology is highlighted in Figure 9.

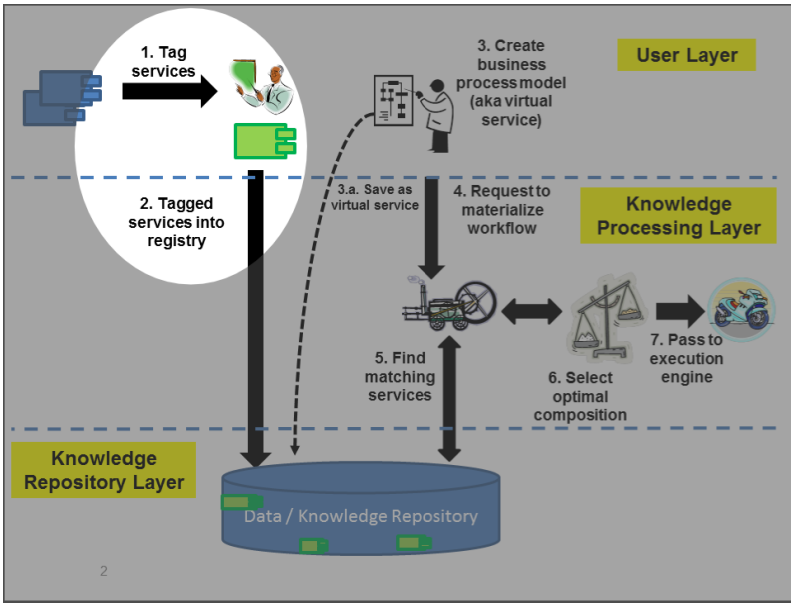

Figure 9: Applying QoS metrics

This research assumes that all web services are implemented as intelligent agents, or that an intelligent agent acts as a proxy for every web service. In this scenario, web

services are not invoked directly, but the input data is passed to an agent that invokes the service and returns the results. This places all services, both web services and physical services, on an equal footing: Any service that is implemented by a person is a service whose proxy is an intelligent agent. For example, a plumber is not the act of fixing a leak; the plumber is the agent who performs that act. The important decision from the user's point of view is to select the best plumber to do the job.

If the agents representing services are empowered to negotiate terms of service (as a human agent is likewise capable of doing), then the playing field among different types of services is leveled still further, and it is possible to implement a more robust negotiation of the QoS for a specific service invocation than is possible with the INRIA model. For example, the FIPA Contract Net Interaction Protocol (50) can be used to negotiate terms of service between a service agent and a broker, or the FIPA English Auction or FIPA Dutch Auction Interaction Protocol can be used to manage bidding among multiple service agents.

The INRIA model also did not include any notion of measuring the quality of service across a service composition or business process. When evaluated across a composition of multiple services, QoS can be measured in terms of whether the composition completes the entire business process or only some fraction of it. For example, let us assume the original process specification includes five distinct activities but it is only possible to find services that complete the first four activities in the process. Despite being incomplete, this partial materialization of the original process specification may still be of some utility to the user. The INRIA model assumes every service

composition completes the process and does not consider a partial process as a potential solution.

Another aspect of assessing the QoS across a service composition is to aggregate each of the QoS attributes of all of the services in a composition and compare those across different composition. There may be many different ways of aggregating different QoS metrics: aggregating cost is a simple summation of the cost of each service in the composition, but aggregating the time required to complets a composition will be more complex if there are services executing in parallel (see Definition 6 for an example of such aggregation).

To meet the needs of evaluating the QoS of a complete service composition, and also to describe QoS of non-web services, this research builds on the INRIA model to define two quality groups that measure service and process QoS: Task and Process.

The Task QoS expresses the quality of service of a single service offering, While analogous to the *Service QoS* in the INRIA model, this expression is more generic to better describe non-web services. This measure serves two different but related functions. It can be applied to an individual service offering to describe the QoS promised by the vendor, and it can also be applied to a task within a process model to express the desired QoS attributes that the modeler thinks are important to the successful completion of the task as a part of the overall business process.

The Process QoS expresses the quality of service of the process as a whole (i.e., the combination of services that fulfills the business requirement of the process). This is

not simply an aggregation of the QoS measurements of the constituent tasks or services, but is a separate measure that relates to the process as a whole.

The structure and scope of these quality groups is described in more detail below.

Task Quality Group

The Task Quality Group describes QoS measures that apply to a single task or activity within a process model (these are the individual blocks within a BPMN model), or to the services that can fulfill a given task. When applied to a task, the modeler will specify the QoS desired for a particular task as part of the modeling process. When applied to a service, the QoS metrics are part of the service description and measure the expected performance of a given service offering.

The Task Quality Group is composed of several quality factors, each with one or more properties as described below.

Quality Factor: Cost

The Cost Quality Factor represents the monetary value of completing a task or invoking a service. The Cost Quality Factor is composed of two properties, Price and Unit.

Property: Price

Price is the amount a modeler is willing to pay to complete an individual occurrence of a task (when applied as part of a process model) or the amount a service provider charges for an individual invocation of a service (when applied to a service offering).

If each service has an agent as its proxy, then the Price property may represent a starting point for negotiations where the agent is empowered to negotiate a final price though processes such as bidding or auctions.

Property: Unit

Unit is the denomination of the Price property, for example dollars or yen.

Quality Factor: Speed

The Speed Quality Factor conveys the maximum time to complete a particular task when applied to the process model, or the time a service is projected to take to complete.

Property: Time

The Time property is the numerical measure of the maximum execution period of the task or of the expected completion schedule of the service.

Property: Measure

The Measure property expresses the units of measure that the Time property captures. For example, Measure may be milliseconds for a web service or days a plumbing service.

Quality Factor: Semantic Similarity

The Semantic Similarity Quality Factor is a calculated metric that embodies the degree to which a candidate service (i.e., a service that may fulfill a given task) matches the semantics of the task. The semantics of a service or a task may be measured along four main facets: inputs, outputs, preconditions, and effects (IOPE). The semantic similarity factor is a measure of the similarity of a service to a task as measured along

each of those four facets. (This is the same way the OWL-S (6) service description captures the elements needed for matching services.)

Property: Effect

The Effect property is a measure of the extent to which the activity's task type as expressed in the process model match a candidate service's task type (i.e,. the service's effect semantics). This measure is a value of the range {0..1}, where 0 indicates no documented match between the semantics of the activity task type and the service task type, and 1 indicates that the activity and the service task types are described using the same ontology. Intermediate values indicate the semantic distance between the activity task type and service task type as measured by the number of ontologies that must be linked to span from one to the other. For example, assume that Activity A is described using Ontology A and Service 1 is described using Ontology 1. If the task type of Activity A explicitly defines the task type of Service 1 as equivalent, then that is a higher-ranking match than the case where the equivalence is established by Ontology A and Ontology 1 both referencing some third ontology.

To better illustrate this idea, assume an Activity and a Service where the task type is described by Attribute X in Ontology A and the service description also describes its task type by referring to Attribute X in Ontology A. In this case, the match is perfect and the Effect property has a value of 1. If the Service description is changed such that its task type refers to Attribute Y in Ontology A, and Attribute Y includes an assertion that it is equivalent to Attribute X, then the match is very close but not quite perfect, and so the Effect property will be something close to, but less than, 1 (perhaps 0.9). If the Service

description is further changed such that its task type refers to Attribute Z in Ontology B, and Attribute Z is asserted to be equivalent to Attribute X, then the value of the Effect property will be still lower, perhaps 0.5. (The assertion of equivalence among attributes within different ontologies may be explicitly defined by the ontologist.) This sort of indirect equivalence could be several ontologies deep, with each link between ontologies resulting in a lower value for the Effect property. For example, assume an activity is described by a task type in a Alice's Weather Ontology. If a service is also described by a task type in Alice's Weather Ontology, there is an exact match. But consider the case where the service is described by a task type in Bob's Meteorology Ontology. If equivalence has been asserted between the task types in each ontology we can still match the service and activity task types, but that match is not as direct as if the service and activity had been described using the same ontology. This sort of indirect matchin coule stretch across several ontologies, with each ontology in the chain decreasing the exactness of the match. For this reason, in practice the number of inter-ontology links should be limited to a relatively small number.

Property: Input

The Input property is an aggregation of the measures of semantic similarity of the inputs of an activity and a service (where the inputs are explicitly captured in the process model). The individual input similarity measures are averaged into a single value within the range {0..1}.

Property: Output

The Output property is an aggregation of the measures of semantic similarity of the outputs of an activity and a service (where the outputs are explicitly captured in the process model). The individual output similarity measures are aggregated into a single value of the range {0..1} that averages the semantic similarity of all output parameters.

Quality Factor: Rating

The Rating Quality Factor measures the reputation of the service offering as measured by users rating their satisfaction with the service.

Property: Ranking

The Ranking property is a value that expresses users' rating of the service offering on whatever scale the individual rating organization employs.

Property: Scale

The Scale property is an integer value that expresses the maximum value a service can achieve on the rating organization's system (e.g., if a system allows users to rank services from 1-5 the Scale would be 5). This value makes it easier to normalize service rankings across different scales (e.g., ratings of 1-4 vs. 1-5).

Quality Factor: Service Type

The Service Type Quality Factor is a measure of whether a candidate service is of the same type as that requested by the process modeler. For example, a process modeler may prefer that all services in a composition be SOAP web services, but some service offerings that otherwise fulfill the task are REST web services.

Property: Interface Type

The Interface Type property is an expression of the way a user invokes the service or a modeler's preferred interface type. For example, one hotel reservation service may offer a REST interface where another only offers a telephone interface (i.e., the user must call the hotel to reserve a room).

Property: Delivery Type

The Delivery Type property expresses the means by which the service is delivered, or the way in which its effects become visible. For example, one hotel reservation service may deliver a confirmation via e-mail where another hotel reservation service delivers confirmation by letter.

Process Quality Group

The Process Quality Group encompasses those attributes of QoS that apply to the process as a whole (when applied to a process model), or to a service composition. When applied to a service composition, the Process QoS measure reflects the characteristics of the aggregation of services and not a summation of the QoS measures of the constituent services.

Quality Factor: Unity

The Unity quality factor is a measure of the number of different service providers involved in a given process or composition. When applied to a process model, the Unity factor expresses the user's preference for minimizing the number of service providers, and when applied to a service composition the Unity factor is a measure of the ratio of service providers involved in the composition relative to the number of services. As an example, some compositions may be assembled from a number of services where each

service is offered by different provider while other compositions may be assembled from a suite of services offered by the same provider. An intermediate situation could be where a group of services are offered by two or more vendors working cooperatively to offer a package of services (as a sort of consortium).

Property: Provider Quantity

The Provider Quantity property is the number of individual service offerors involved in a given service composition. For example, a composition composed of three services all offered by the same company would have a Provider Quantity of 1.

Property: Service Quantity

The Service Quantity property is the number of service invocations required to complete a given service composition.

5.2.QoS Calculation

The QoS of a Virtual Service Instance (VSI) is the basis of the optimization assessment defined in Chapter 4. This section provides a brief intuitive explanation of how each of the QoS metrics described in the formalism is aggregated for a VSI. Optimization based on the QoS metrics that are not discussed in the formalism in Chapter 4 remains an area for future research.

The QoS of a VSI is calculated by evaluating the QoS of each atomic service and aggregating those measures accordin to a defined formula. In many cases, this aggregation may be a simple averaging of a given QoS metric across all atomic services, but in other cases the aggregation may be more complex such as that shown in Definition 7. (As specified above, the atomic service is the level at which QoS is defined.) However,

aggregating the QoS measures is not a simple matter of summing the QoS measures of each atomic service in a VSI. The specific calculations for aggregating QoS across a VSI are described below.

Cost

For a given VSI, the cost of the VSI is calculated by summing the costs of each atomic service that is used in that VSI. While this definition may seem intuitive, its development is helpful in understanding how other QoS measures are calculated. The formal definition of the cost of a VSI is provided in Definition 5.

Duration

The duration of service execution, when aggregated across all atomic services within a VSI, yields the expected duration of the VSI. Calculating the duration of a VSI is somewhat more complex than calculating cost because it is not a simple summation of the durations of each atomic service invocation. Some services may be executed in parallel. If one service takes longer to complete than the other, the shorter service's duration is not a factor in calculating the overall duration of the service composition because the longer service will still be executing after the shorter service has completed.

Given this potential parallel execution of services, it is necessary to calculate the longest duration path through the service composition, based on the duration of each atomic service. The shortest time in which the composition can be executed is thus the sum of all the longest durations of each set of parallel services.

The formal definition of this calculation is provided in Definition 6, and is explained intuitively as follows. The VSI is divided into "phases," where the first atomic

service is in Phase 1, all atomic services that are called using the outputs of the atomic service in Phase 1 are in Phase 2, and so forth. Those services in Phase n are those that require the outputs of services in Phase n-1. Using this construct, we calculate the longest duration of a service composition by adding the duration of the service in Phase n (i.e., the final atomic service) to the duration of the longest service in Phase n-1, adding this total to the duration of the longest service in Phase n-2, etc. This is required to account for the cases where multiple services are executing in parallel (i.e., during the same phase) and some of those services take longer than others to complete; the overall process cannot proceed to the next phase until all services in the current phase have completed. The resulting total is the minimum amount of time it will take the full composition to complete.

Rating

The rating of a VSI is formally defined in Definition 7 and it intuitively explained here. Each atomic service has a rating that measures users' reported satisfaction with that service, together with the scale the user rating is measured on. For example, a user may have rated a service as a four on a scale of one to five.

All service ratings are normalized by dividing the user rating by the maximum possible rating, resulting in a value in the range [0..1]. After all of the services' ratings are normalized, the ratings for all services in that VSI are averaged to compute the rating for that VSI.

6. EXTENSIONS TO BPMN

As discussed in Section 2.2, existing process specification modeling languages do not contain the semantic information necessary to support service-to-activity matchmaking. Previous efforts were directed towards service descriptions based on SOAP-described web services.

In contrast, this research develops a more generic approach that supports matchmaking of activities to services described using semantic markup. This motivated the development of an extension to BPMN, called BPMN-S, for the semantic markup of process models.

The extensions to BPMN are used in the portion of the methodology highlighted in Figure 10.

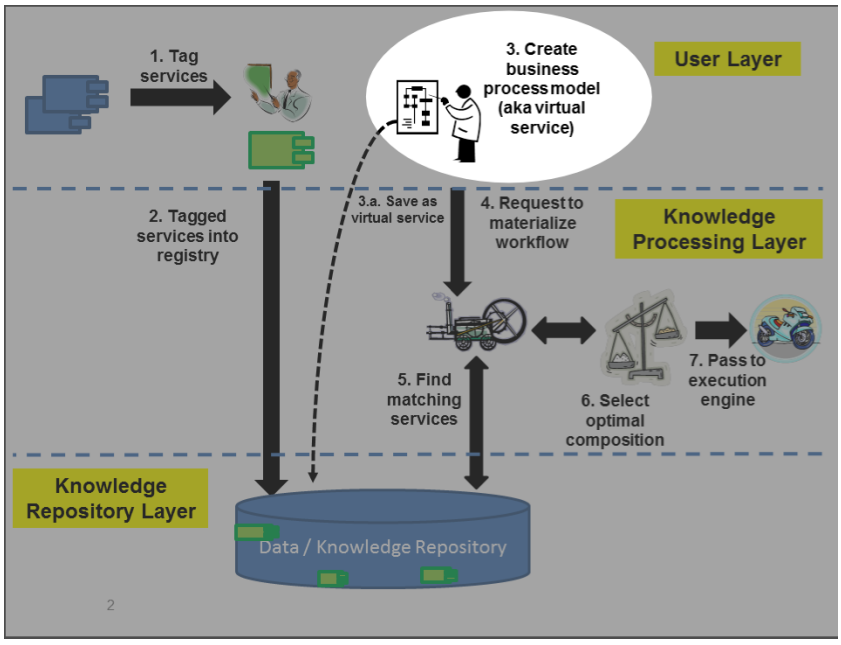

Figure 10: Using BPMN extensions

6.1. Language Selection

One option for enabling the semantic markup of process models would have been to create a new modeling language or notation; this is essentially the approach taken by the WSMO project as described in Section 2.2.3. However, the development of a full-featured process modeling language from scratch is beyond the scope of this research. Therefore, our approach has been to build on an existing language, BPMN, and extend it to meet the needs of this research.

Existing process description models fall into two broad categories:

- Graphical languages intended for use by business analysts or system designers such as BPMN and Unified Modeling Language UML (52),
- Procedural languages intended for use by developers such as BPEL.

Because a goal of this research is to enable business analysts to specify process models that can be matched to services, procedural languages are not suitable for specifying a process. Of the graphical languages, BPMN was selected as the modeling language for this research.

BPMN is a graphical language that is readily understandable by business analysts and can be learned by end users in a relatively short time. In addition, UML activity diagrams are specifically designed to model the process that a software module executes. However, as discussed in Section 2.2.4, UML's lack of a standardized machine-readable representation such as XML, combined with inherent ambiguities in the language, make it unsuitable as the basis for an automated service composition language. The XML Metadata Interchange (XMI) format published by the Object Management Group (OMG) is machine-readable, but its purpose is to describe the appearance of a UML model and not the significance of the model elements (e.g., the differences between a process activity and a data artifact).

BPMN is a graphical modeling language intended for use in describing business processes from the end user's point of view. Additionally, since the release of BPMN 2.0, BPMN has a formal XML notation that makes BPMN models machine-readable. While BPMN models can be ambiguous if the modeler is not careful, applying the principles described by zur Mehlan in (53,54) reduces ambiguity in BPMN models..

6.2. Language Extension

After selecting BPMN as the foundation for an enhanced process modeling language, this research investigates how BPMN can be extended to support automated service composition.

Extending the XML notation of BPMN would necessitate modifying a modeling tool to export the new XML elements and attributes as part of converting the BPMN model to XML. Rather than extend the BPMN XML specification, this research captures the semantic information in the documentation field of each of the activities in the model. This ensures the information is exported as part of the normal XML generation process. This eliminates the need to modify the modeling tool and ensures compatibility with the BPMN 2.0 XML schema. The specific documentation annotations are explained in detail below.

6.2.1. Activity Semantics

The first task is to establish the information elements necessary to allow automated service-to-activity matchmaking. For the matchmaking to be effective, each activity in the process model must be annotated with a semantic description that can be matched to the semantic descriptions of service operations. This can be accomplished through the use of a simple list of allowable values, but that is essentially a syntactic matching scheme that is only as flexible as the list of values.

A better solution than a list of values is the use of references to one or more external ontologies to define the semantics of the activities in the process. This has several advantages. First, it allows a business process modeler to refer to an ontology separate from the process model, ensuring each can evolve independently, thereby

promoting reuse of existing ontologies. Second, ontological references at the activity level allow the use of multiple ontologies within a single process model, increasing flexibility.

To accommodate references to external ontologies, it is necessary to add an ontology annotation to each of the activities within the process model, along with a notation denoting the specific task type in the ontology that this activity refers to. Within the documentation field, the ontology reference is documented in the following format:

`ontology:<ontology_URI>`

This notation specifies the ontology that is used to define the semantics used for this activity and its inputs and outputs. The task type is denoted as follows:

`type:<task_type_reference>`

The task type is an entry in the ontology specified by the ontology reference. If this ontology is the same as the ontology used to generate the service descriptions, then service-to-activity matchmaking is a simple matter of matching the effect of an operation in the service description to the task type. In the event different ontologies are used for the process model specification and the service description, there are several possible means of establishing equivalence between the different terms. This could be done through an assertion on a term in one ontology, to which a particular term in the other ontology is equivalent, or it could be through the sort of automated ontology matching described by Muthaiyah et al (51).

6.2.2. Input and Output Parameters

In addition to specifying activity semantics, it may be desirable to specify the semantics of the input and output parameters of each activity in the model. These parameters must refer to the same ontology as the activity type reference, with the same benefits that such a reference provides for activity semantics.

Input semantic types for each activity in the model are captured in the documentation field using the following notation:

`input:<input_type>`

There is an individual entry for each of the inputs to an activity. Each input is captured on a separate line to simplify the task of parsing the inputs out of the documentation field.

Output semantic types for each activity are captured the same way the input types are:

`output:<output_type>`

As is the case for inputs, there can be many output parameters and each one is captured on a separate line within the documentation field for the activity.

Adding input and output data elements to each activity in the model enables refinement of the selection of services, by ensuring the resulting service selections can be composed into a complete workflow; every service selected for a given activity will have the same inputs and outputs.

7. OGMA: A SERVICE DESCRIPTION LANGUAGE

The purpose of a service description is to capture the information necessary to understand how to bind to and invoke the service. As discussed in detail in Section 2.1, none of the existing service description models includes sufficient semantic detail to enable automated activity-to-service or service-to-service matchmaking. The remainder of this section describes the OGMA service description language developed by this research to address the limitations of current service description models.

The service description language is applied during the service tagging and registration part of the DRUID methodology, as highlighted in Figure 11.

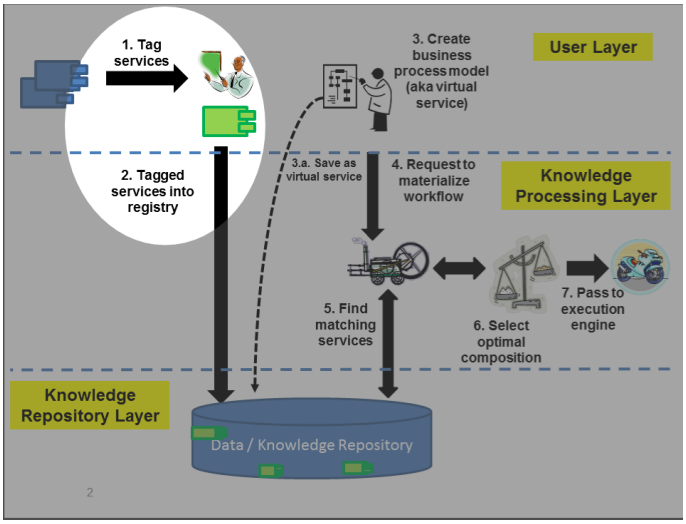

Figure 11: Creating service descriptions

7.1. Design Challenges

To overcome the limitations of existing service interface description models, this research defines a service description language that fulfills the following key criteria:

- Support for descriptions of many service types: SOAP services, REST services, physical services, and services that do not fit into one of these categories,
- Provides the necessary semantic markup to enable the matchmaker to perform service-to-activity matchmaking,
- Facilitate service-to-service matchmaking based on semantic descriptions of service inputs and outputs.

Additionally, the service description format should include all the elements of both WSDL and WADL service descriptions, so as to simplify converting existing service descriptions to this new format. Given the large number of existing WSDL service descriptions, and to a much lesser extent WADL descriptions, this type of compatibility is an important practical consideration.

Before discussing the design of the OGMA language in detail, it is helpful to define semantics within the context of a service description and its importance to automated matchmaking.

7.1.1. Semantics Defined

In general, semantics is the study of the meanings of words. For purposes of this discussion, "semantics" refers to the explicit encoding of the meaning of terms within a service interface description. This includes the meaning of individual data elements that

are the inputs or outputs of the service, the function of the service (i.e., the task it performs), and the meaning of additional useful information.

Semantics may be formally encoded using any of several ontology definition languages. One of the most common is the Web Ontology Language (OWL) (55). OWL is actually a family of languages that can be used to formally describe a body of knowledge, including the entities within that body, their characteristics, and the relationships among those entities and characteristics. OWL is based on Description Logics (56) and the Resource Description Framework (RDF) (57). RDF assertions are constructed as triples that take the form subject-predicate-object (e.g., Person hasName Bob). RDF triples, and by extension OWL, can be combined to explicitly define any concept.

When semantics are encoded in RDF or any of its derivatives, the formal structure of the encoding enables rule processing and basic machine processing of the contents of the ontology. Given a sufficiently detailed ontology, this processing can determine if two different terms refer to the same concept. For example, let us say the term “position” is defined as having attributes “latitude” and “longitude.” We can then infer that the term “location” with attributes “latitude” and “longitude,” refers to the same concept as “position.” Some reasoners, such as HermiT, can infer this equivalence without additional information. Other reasonsers require the addition of specific equivalence rules using the Semantic Web Rule Language (SWRL) (58). To some extent, this processing is possible even when different ontologies are used by different parties, as described by Muthaiyah et al in (51,59,60).

Semantics can also be encoded in a less formal manner through the use of controlled vocabularies. A controlled vocabulary is a limited set of terms used by a community of interest to encode data within that community. A controlled vocabulary need not be formally encoded in any way; all that is necessary is that the terms be limited and agreed upon. However, the use of controlled vocabularies is very limiting because it does not support the automated reasoning or inference enabled by OWL.

7.1.2. Data Semantics

Regardless of the type of service or the complexity of the data it consumes or produces, the inputs and outputs of a service can be broken down into several atomic elements. Each of these elements has its own semantics that must be understood by the consumer before the service can be correctly invoked. Consider a service that returns the current temperature for a given location. This service takes one input parameter named "location" of type "string" and returns one output parameter named "temperature" of type "integer." If this is all the information available it will be difficult to successfully invoke the service. The input parameter "location" may be any of several location designators. It may be a city name, a postal code, a Universal Transverse Mercator (UTM) coordinate, or some other location designator.

Changing the name of the input parameter name to "postal_code" does not eliminate this ambiguity. We still do not have enough semantic information to invoke this service because postal codes are not standardized across the world. In the UK, postal codes are called "postcodes" and are an alphanumeric designator composed of between six and eight characters including a single space. In the US, postal codes are referred to as

Zone Improvement Plan (ZIP) codes and are a numeric designator of five digits. Thus, we can see that "postal_code" by itself does not convey enough semantics to successfully invoke the service.

In order to understand the semantics of the data accurately enough to invoke the service, the semantics must be explicitly described and linked to the service description. To automatically process the inputs and outputs of a service, the semantics of the data elements must be in a machine-readable format linked to a machine-readable service description.

7.1.3. Operation Semantics

In addition to data semantics, it is also necessary to understand the functions a service performs on that data. The weather service described earlier accepts "postal_code" as input and returns "temperature" as output. Let us assume that an operation offered by that service is called "getTemperature." The name alone is insufficient to understand the operation. The operation may return the current temperature for the given location, it may return the maximum forecast temperature, or it may return any of several other temperature readings for that location.

Using more explicit operation names does not remedy the ambiguity. Just as data elements must include explicit semantic references, operations within a service description must also include explicit semantic references. As is the case for data semantics, the semantic references for service operations should be machine-readable and explicitly linked to the operation name.

7.1.4. Service Semantics

The semantic description of a service is defined as the task type performed by the primary operation the service provides. Because a service is a collection of operations, the semantics of the service are relatively unimportant if the operations are semantically tagged. However, semantically tagging the service is helpful because it can simplify searching for operations by providing a ready means for categorizing services.

7.2. Model Definition

Because of the need for clear semantics as described in Section 7.1.1, this research uses OWL to define the OGMA service description language. OWL has the added advantage of enabling machine reasoning across the service descriptions.

The OGMA service description language is described in detail in the paragraphs that follow. The OWL specification of the language can be found in Appendix A. The service description language is illustrated by the use of an example service. The example service is one that reserves a hotel room. This example is loosely based on the hotel reservation services offered by Marriott.

The foundation of the service description language is the SERVICE class, which is defined as an aggregate of several constituent classes that define the details of the operations offered by the service. The service class includes a name for the service and an industry classification code for categorizing the business domain the service is intended to serve. A description of the SERVICE class is shown in Table 17.

Table 17: Service Definition

Service Definition
SERVICE has attributes [name string] [naicsCode string] is aggregate of [class BINDING] [class OPERATION] [class PROVIDER] [class RESOURCE] [class STATE]

Applying this template to Marriott yields the description shown in Table 18, where the NAICS code 561599 designates "All Other Travel Arrangement and Reservation Services."

Table 18: Marriott Service Description

Attribute	Value
Name	All Other Travel Arrangement and Reservation Services
naicsCode	561599

The BINDING class is an abstract class that is further subdivided into subclasses NEWORKBINDING and PHYSICALBINDING. These classes encapsulate the information needed to invoke an individual OPERATION and are described in Table 19.

Table 19: Binding Definition

Binding Definition	
BINDING is *abstract*	
[name	string]
NETWORKBINDING extends BINDING	
[communicationProtocol	{http|https|sms|smtp}]
[soapProtocol	{document|rpc}]
[url	URL]
[soapTransport	URI]
PHYSICALBINDING extends BINDING	
[city	string]
[state	string]
[country	string]
[communicationProtocol	{person|postal|telephone}]
[phoneNumber	string]
[pointOfContact	string]
[postalCode	string]
[address	string]

The NETWORKBINDING class has the attributes necessary to bind to services that are accessible via the Internet. The communicationProtocol attribute takes one of several attributes as listed to define whether the service is bound using HTTP or HTTPS in the case of SOAP or REST services, or Short Message Service (SMS) for services that may be accessible by a cell phone text message. It also includes Simple Mail Transfer Protocol for services that may be invoked using e-mail.

The PHYSICALBINDING class has those attributes that are needed to invoke a physical service, whether in person or by contacting a person. The communicationProtocol attribute defines the means by which the service provider is contacted, either in person, by mail, or by telephone. The remainder of the attributes are

standard elements used to define the physical location of a place of business, such as a street address, city, etc.

Continuing to describe Marriott with the service description language, we include both network and physical bindings for operations the Marriott service offers. The network binding, shown in Table 20, defines the information needed to connect to Marriott's web site. The physical binding, also shown in Table 20, includes the information needed to contact the Marriott reservation service person-to-person.

Table 20: Marriott's Binding

Network Binding	
Attribute	Value
communicationProtocol	http, https
soapProtocol	null
url	http://www.marriott.com
soapTransport	null
Physical Binding	
Name	Marriott Hotels Binding
City	Bethesda
State	MD
Country	USA
communicationProtocol	telephone
phoneNumber	1-888-236-2427
pointOfContact	Reservations
postalCode	20817
Address	10400 Fernwood Rd

The PROVIDER class defines the person or organization that offers the service being described. The PROVIDER class is depicted in Table 21. Each of the attributes of the PROVIDER class is a common attribute of any business or personal contact information and is not described in any additional detail here.

Table 21: Provider Definition

Provider Definition
PROVIDER [city string] [state string] [country string] [identifier string] [name string] [phoneNumber string] [postalCode string] [address string] [url URL]

In the case of Marriott, the PROVIDER information is shown in Table 22, which depicts the information for the business entity that has overall responsibility for provisioning the services on offer.

Table 22: Provider Information

Attribute	Value
City	Bethesda
State	MD
Country	USA
Identifier	
Name	Marriott Hotels, Inc.

phoneNumber	1-301-380-7770
postalCode	20817
Address	10400 Fernwood Rd
url	http://www.marriott.com

The RESOURCE class, defined in Table 23, describes the input or output of any service. The RESOURCE class itself is abstract and contains three attributes. The first is currentState, which is a reference to a STATE object that describes the current state of the resource (the STATE class is described below). Each RESOURCE is identified by a name attribute and includes an ontology attribute that references a URI that formally describes the semantics of the RESOURCE.

Table 23: Resource Description

Resource Definition
RESOURCE is *abstract* [currentState STATE] [name string] [ontology URI] PHYSICALRESOURCE is *abstract*, extends RESOURCE VIRTUALRESOURCE is *abstract*, extends RESOURCE

The RESOURCE class has two abstract subclasses, PHYSICALRESOURCE and VIRTUALRESOURCE, that describe resources that are either physical objects or data types respectively.

The abstract class PHYSICALRESOURCE has two subclasses, ANIMATERESOURCE and INANIMATERESOURCE, as shown in Table 24. These classes are used to describe physical objects that are inputs or outputs to a service. The ANIMATERESOURCE class describes living things that a service operates on. The ANIMATERESOURCE class has two attributes, genus and species, corresponding to the elements of the same name used in the Linnaean taxonomy (61).

Table 24: PhysicalResource Definition

PhysicalResource Definition
PHYSICALRESOURCE is *abstract*, extends RESOURCE ANIMATERESOURCE extends PHYSICALRESOURCE [genus string] [species string] INANIMATERESOURCE extends PHYSICALRESOURCE [description string]

The class INANIMATERESOURCE, also shown in Table 24, includes one attribute, description, for designating the type of item the service acts upon. To illustrate the application of the RESOURCE classes, Table 25 shows a subset of potential inputs and outputs for a Marriott service.

Table 25: Medical Resources

Inanimate Resource	
Attribute	Value
Description	luggage

In Table 23 it can be seen that the currentState attribute has a type STATE. The STATE class defines the current condition of a RESOURCE. The STATE class is described in Table 26.

Table 26: State Definition

State Definition	
STATE	
[status	string]
[description	string]
[ontology	URI]

The STATE class denotes the status a RESOURCE may assume. The status is stored as a string, with an accompanying description attribute that can provide amplifying information as needed. There is also an ontology attribute that can reference an external ontology that describes the semantics of the STATE being referenced. An example STATE object is shown in Table 27.

Table 27: State Example

State	
Attribute	Value
Status	unconfirmed

State	
Attribute	Value
description	A reservation has been requested but has not yet been verified
ontology	http://myontology.org/hospitality#unconfirmed

This example STATE is for a customer reservation that has been scheduled but has not yet been confirmed (for example, by submitting a credit card as a surety). This would typically be the STATE of a customer reservation that is being processed by the Marriott reservation service.

The OPERATION class describes an individual function offered by a service PROVIDER. OPERATION itself is an abstract class with four concrete subclasses as shown in Table 28.

Table 28: Operation Definition

Operation Definition
OPERATION is *abstract* [name string] [effect STATECHANGE] [input RESOURCE] [output RESOURCE] [binding BINDING] [semanticType URI] [precondition STATE] NOTIFICATION extends OEPRATION ONEWAY extends OPERATION REQUESTRESPONSE extends OPERATION [httpMethod {GET|POST|PUT|DELETE}]

Operation Definition
SOLICITRESPONSE extends OPERATION

The subclasses of the OPERATION class mirror those defined in the WSDL standard in order to ensure compatibility with SOAP-based web services but are also compatible with REST services and physical services. As the definition shows, an OPERATION contains many references to other classes. Each OPERATION is identified by a name attribute that is a string. The action performed by the OPERATION is identified by a reference to a STATECHANGE object (defined below); that action takes some RESOURCE object as an input and returns some RESOURCE object as an output. A given OPERATION may change the STATE of an input RESOURCE and return the altered RESOURCE, while another OPERATION may take one type or RESOURCE as an input and return a different type of RESOURCE as an output.

The information needed to bind to and invoke an OPERATION is encapsulated within a BINDING object. Each OPERATION may be characterized by a semanticType as defined by an external ontology, and it may also define a necessary precondition denoted by some STATE.

The subclass REQUESTRESPONSE includes an additional attribute, httpMethod, denoting one of the four HTTP methods (GET, POST, PUT, or DELETE) that describes the method used to invoke an OPERATION using the HTTP protocol. A notional example OPERATION offered by Marriott is shown in Table 29 (Marriott does not currently offer a publicly available web service for reserving hotel rooms).

Table 29: Operation Example

Operation	
Attribute	Value
name	reserveRoom
effect	confirmedReservation
input	arrivalDate
input	departureDate
output	confirmationNumber
binding	Marriott Hotels Binding
semanticType	http://myontology.org/hospitality#reserve
precondition	Null

The STATECHANGE class referenced in Table 28 is defined in Table 30. The STATECHANGE class is a convenience class that defines an initialState and finalState, encapsulating an effect that could be produced by many different operations.

Table 30: StateChange Definition

StateChange Definition	
STATECHANGE	
[name	string]
[initialState	STATE]
[finalState	STATE]

An example STATECHANGE is shown in Table 31. The example describes a change in state that may be applied to many resources affected by different operations; in

the example service it would apply to the blood that is an input to the bloodAnalysis operation.

Table 31: StateChange Example

StateChange	
Attribute	Value
name	unconfirmedToConfirmed
initialState	Unconfirmed
finalState	Confirmed

The framework employs a main ontology that embodies the service description language explained above and encodes the semantics of the service description. This ontology is encoded in OWL, making it easier to query the service descriptions when they are stored in an OWL-compliant repository.

The SERVICE class is a grouping of operations, and so is related to the OPERATION class as depicted in Table 32.

Table 32: Service Object Properties

SERVICE Object Properties		
Property	Cardinality	Object
hasNAICSCode	some	Integer
hasName	exactly 1	String
hasOperation	some	OPERATION

The SERVICE property hasNAICSCode is a means for describing the business domain of a service using the North American Industrial Classification System code, and has as its object zero or more integers. The property hasName takes exactly one string as its object and provides a means for identifying the SERVICE. The SERVICE property hasOperation takes as its object zero or more objects of the OPERATION class. A service with zero operations has no practical value, but could be used to describe an abstract service.

The object properties of the OPERATION class are shown in Table 33.

Table 33: Operation Properties

OPERATION Object Properties		
Property	Cardinality	Object
hasBinding	some	BINDING
hasEffect	some	STATECHANGE
hasInput	some	RESOURCE
hasOutput	some	RESOURCE
hasPrecondition	some	STATE
hasSemanticType	some	String

The OPERATION property hasBinding has an object of type BINDING. Each OPERATION may have multiple bindings; for example, a web service may offer both REST and SOAP interfaces, and so would require two different bindings. The property hasEffect takes as its object some number of STATECHANGE objects. An individual OPERATION will generally have a single effect, but a composite OPERATION (e.g., one that represents an orchestration of other operations) may have multiple effects.

The hasInput and hasOutput properties both have an object of type RESOURCE. Each OPERATION will accept zero or more inputs and will produce zero or more outputs. A given OPERATION may require some external condition to prevail before it can be invoked, so the hasPrecondition property specifies any STATE that must be true before the service can be executed. The property hasSemanticType is a means for specifying a correspondence between an OPERATION and some defined semantic representation. Ideally this representation would be some element within an externally defined ontology, but it is defined as a simple string to support use of a simple controlled vocabulary that is captured directly in the service descriptions. The hasSemanticType property is the means by which an OPERATION is matched to a BPMN activity during the service-to-activity matchmaking process.

The OPERATION class itself is abstract. Each of its subclasses (NOTIFICATION, ONEWAY, REQUESTRESPONSE, and SOLICITRESPONSE) inherits all these properties and none has additional object properties.

The BINDING class is abstract and has no properties of its own. Its descendants are described in Table 34 and Table 35.

Table 34: Network Binding Properties

NETWORKBINDING Object Properties		
Property	Cardinality	Object
`hasCommunicationProtocol`	`some`	`{"http", "https", "sms", "smtp"}`
`hasSoapStyle`	`max 1`	`{"document", "rpc"}`
`hasSoapTransport`	`max 1`	`String`
`hasURL`	`Some`	`String`

The NETWORKBINDING class describes how to bind to a web service, whether a SOAP service or a REST-style service. The hasCommunicationProtocol element has allowable values that describe the protocol that can be used to connect to the service. For SOAP services, the binding may include at most one hasSoapStyle property with allowable values of "document" or "rpc" depending on how the SOAP service is offered. The hasSoapTransport protocol is defined by at most one string and is also confined to SOAP services. Finally, the hasURL property contains the service endpoint for a web service.

Table 35: Physical Binding Properties

PHYSICALBINDING Object Properties		
Property	Cardinality	Object
hasCity	some	String
hasCommunicationProtocol	some	{"person", "postal", "telephone"}
hasCountry	some	String
hasPhoneNumber	some	String
hasPointOfContact	some	String
hasPostalCode	some	String
hasStateOrProvince	some	String
hasStreetAddress	some	String

The properties of the PHYSICALBINDING class describe how to interact with a physical service and provide contact information including the street address and phone

number for invoking a given operation. Most of the property names are self-explanatory, but two are worth explaining in more detail. The hasCommunicationProtocol has a set of allowable values that describe how a user would interact with a physical operation. The hasPointOfContact property defines the person to coordinate invocation of the operation of that is needed (for example, a plumber may include a receptionist's name as the point of contact for scheduling service).

The properties of a PROVIDER object are described in Table 36. Many of these properties are identical to those found in the PHYSICALBINDING class and are equally self-explanatory.

Table 36: Provider Object Properties

PROVIDER Object Properties		
Property	Cardinality	Object
hasCity	some	String
hasCountry	some	String
hasIdentifier	some	String
hasName	some	Literal
hasPhoneNumber	some	String
hasPostalCode	some	String
hasStateOrProvince	some	String
hasStreetAddress	some	String
hasURL	some	String

Several properties of the PROVIDER class require a more explicit definition. The hasIdentifier property is used to specify the unique identifier of this provider in, for example, a social media system or other registry where information about that provider

can be retrieved and used as part of the process of developing service recommendations (as described later). The hasName property specifies the common name by which the provider is identified (e.g., a business name). The hasURL property offers an opportunity for a service provider to include a web site or other network identifier (e.g., a public Facebook page).

The properties of the RESOURCE object are described in Table 37. While the RESOURCE class itself is abstract, these properties are common to its subclasses.

Table 37: Resource Object Properties

RESOURCE Object Properties		
Property	Cardinality	Object
`hasCurrentState`	`some`	`STATE`
`hasName`	`some`	`String`
`isDescribedByOntology`	`some`	`anyURI`

The hasCurrentState property takes an object of type STATE and defines the state of a given resource at any given time. The hasName property offers the option of defining a convenient identifier for a RESOURCE, and the isDescribedByOntology offers the opportunity to specify an external ontology that describes the RESOURCE in more detail.

The PHYSICALRESOURCE class is an abstract subclass of the RESOURCE class with no distinct properties of its own. The subclasses of the PHYSICALRESOURCE class are ANIMATE and INANIMATE and their properties are described in Table 38 and Table 39.

Table 38: Animate Object Properties

ANIMATE Object Properties		
Property	Cardinality	Object
`hasGenus`	`some`	`String`
`hasSpecies`	`some`	`String`

The ANIMATE class describes a RESOURCE that is a living thing that may be the input or output to an operation. Its two properties, hasGenus and hasSpecies, define the genus and species of any living thing as used in biology.

Table 39: Inanimate Object Properties

INANIMATE Object Properties		
Property	Cardinality	Object
`hasDescription`	`some`	`String`

The INANIMATE class describes non-living physical objects and has a single property, hasDescription, that provides some means of identifying the object (this identifier should be related to the isDescribedByOntology property inherited from the RESOURCE class).

The VIRTUAL resource class is used to describe operation inputs and outputs that are information that can be transported across the network, and borrows heavily from the data definitions of both SOAP and REST-style services. Its properties are described in Table 40.

Table 40: Virtual Resource Object Properties

VIRTUAL Object Properties		
Property	Cardinality	Object
hasContentType	max 1	String
hasContentSubType	max 1	String

The hasContentType and hasContentSubType of the VIRTUAL resource class allow a service provider to describe an input or output data type that does not lend itself to the more precise descriptions offered by the ELEMENT and MESSAGE classes described below.

The ELEMENT class is derived directly from the element definition within the WSDL specification and is described in Table 41.

Table 41: Element Resource Object Properties

ELEMENT Object Properties		
Property	Cardinality	Object
hasDataType	exactly 1	String
hasMaxCardinality	exactly 1	Integer
hasMinCardinality	exactly 1	Integer
hasName	some	Literal
hasSemanticType	some	String

The hasDataType property refers to the XML data type of the element. The hasMaxCardinality and hasMinCardinality properties define, respectively, how many or

how few of the elements may be present. The hasName property provides a means of identifying the element in a human-readable form, and the hasSemanticType property provides a means for associating the ELEMENT with an externally defined ontology.

Table 42 describes the properties of the MESSAGE virtual resource. Like the ELEMENT resource, the MESSAGE resource is derived directly from the message definition in the WSDL specification.

Table 42: Message Resource Object Properties

MESSAGE Object Properties		
Property	Cardinality	Object
hasElement	only	ELEMENT
hasElement	some	ELEMENT

The MESSAGE resource has only one property, hasElement, with two separate restrictions: the hasElement property can only be populated by objects of the type ELEMENT, and it can have any number of ELEMENT objects.

The STATE object describes the state of any RESOURCE and its properties are shown in Table 43.

Table 43: State Object Properties

STATE Object Properties		
Property	Cardinality	Object
hasDescription	some	String
hasStatus	exactly 1	String
isDescribedByOntology	some	anyURI

The hasDescription property provides a place to include an informal description of the STATE of the RESOURCE, and is a companion to the hasStatus property, which is a more formal definition of the STATE. The isDescribedByOntology property provides an option for linking this STATE to an ontology that describes it more thoroughly.

The STATECHANGE class describes the effect an OPERATION has. Its properties are described in Table 44.

Table 44: StateChange Object Properties

STATECHANGE Object Properties		
Property	Cardinality	Object
`hasFinalState`	`some`	`STATE`
`hasInitialState`	`some`	`STATE`

The hasFinalState and hasInitialState properties each take an object of type STATE, and taken together define the transition from one STATE to another STATE that an OPERATION effects.

While storing service descriptions in an ontology makes it possible to reason over them, the OGMA language enables additional reasoning through the use of auxiliary ontologies used to describe the operations offered by the services and the resources those operations use as inputs and outputs. Each attribute that may require a semantic reference (in a case where the set of services being described does not use a controlled vocabulary)

is defined by a URI that references the corresponding entity within one or more ontologies.

The use of externally-defined ontological descriptions for operations and resources eliminates the need for different service providers to standardize on a common ontology, thereby simplifying the task of combining operations from different providers to compose a workflow from operations offered by different service providers.

7.3. General Service Descriptions

The key challenge in developing the OGMA service description language is describing all types of services within a single description format. WSDL and WADL each do a reasonably job of describing SOAP and REST services respectively, and WSDL 2.0 attempts to combine both SOAP- and REST-based descriptions into a single format. But no service description model has attempted to describe services as diverse as a SOAP-based weather service, a REST-based hotel reservation service, and a physical package delivery service within the same language.

All of the elements of both the WSDL and WADL models are retained to simplify the conversion of existing service descriptions to the OGMA format,. In addition, the information needed when invoking a physical service had to be included, as well as information suitable for invoking services that may not fall into one of those categories or that may span categories. For example, a hotel may offer a reservation service in any or all of the following ways:

- As a SOAP web service that can be invoked from an application,
- As an e-mail based service where a user can e-mail a reservation request,

- As a telephone-based service where a user speaks to a reservation agent.

In all of these cases, the reservation is ultimately entered into and managed via the hotel's reservation management system. The service is the same, only the interface is different.

To accommodate this variety of possible interfaces, the OGMA language expands on earlier concepts of service interface to accommodate person-to-person and person-to-machine interactions. The OGMA language extends the concept of a "binding" from the WSDL description to include more detailed means of specifying how a user invokes a service endpoint.

A general service description should include the ability to specify the physical inputs and outputs of a service for those services that interact with the physical world. For example, an auto mechanic offers a service that repairs cars; one of the outputs of this service is a repaired car. To accommodate this, the OGMA language extends the concept of a "resource" as defined by the WADL format, expanding "resource" to include both physical and virtual resources so that the inputs and outputs of a service can be thoroughly defined.

The details of the definition of "binding," "resource," and other aspects of the OGMA language are explain in Section 7.2.

7.4. Activity-to-Service Matchmaking

A good service description must support matching individual service operations to each activity in a process model. This matchmaking requires that both the service description and the activities in the process model contain sufficient semantic information

to determine the service and activity refer to the same type of work. Matchmaking may also require that the service and activity have compatible input and output parameters.

When the semantic annotations of service operations and activities are taken from the same ontology, the matchmaking process is a simple matter of matching identical semantic tags. Where the semantic annotations are based on different ontologies, there must be some means for asserting the equivalence of terms in different ontologies. This equivalence may be asserted by a business analyst or it may be the result of an automated process such as the use of the Semantic Web Rule Language (SWRL) as described in (51).

Regardless of whether the process model and services use a single ontology or multiple ontologies, matching services to activities in the process model is the first step in specifying an executable service composition. This initial screening of candidate services selects those services, that correspond to an activity in the process model, for subsequent analysis to see if they can be composed to materialize the process.

7.5. Service-to-Service Matchmaking

Service-to-service matchmaking is the second phase of the matchmaking process. Once the available services have been filtered to those corresponding to activities in the process model, it is necessary to determine which services are composable. This step entails comparing the semantics of input and output parameters of each of the matched services. For two services to be composable all of the required inputs of one service must be among the inputs to, or outputs from, preceding services in the composition. For example, if we with to compose services A and B together, then all the required inputs to

Service B must be available among the inputs and outputs of Service A. This constraint if formally defined in Definition 3.

Just as for activity-to-service matchmaking, the semantic annotations for input and output parameters may be captured in a single ontology or in multiple ontologies. If the semantics are captured in multiple ontologies, there must be some means of asserting the equivalence of semantic annotations in different ontologies.

8. PROOF OF CONCEPT PROTOTYPE

In order to demonstrate the feasibility and utility of the contributions of this research, a proof-of-concept prototype was developed to exercise the components of the framework and to ensure they performed as expected. Some of the original objectives of this research were modified as a result of lessons learned during the implementation of this prototype. For example, the initial objective of this research was to automate the composition of web services. However, after analyzing a wide variety of business process models, it became clear that nearly all practical business processes involve some level of human involvement, whether to perform a sophisticated analysis step, or to approve final payment of an invoice. This realization resulted in an expansion of this research to include physical services as well as web services.

8.1. Design

One of the primary design objectives, when developing the proof-of-concept prototype, was to reuse existing tools whenever possible. A key benefit of the SOA concept is the reuse of existing services, so it seems fitting to reuse existing tools when building a prototype intended to demonstrate the benefits of service reuse.

Another design consideration was to use as many established standards as possible when constructing the prototype. Some standards, such as BPMN, are extended as a key part of this research, but by leveraging other standards it is hoped that the

success of this prototype will be seen as a validation of the key contributions of this research, and not as a demonstration of particular programming techniques.

By the same token, this prototype is not intended to be a fully operational system that could be deployed in a production environment. Accordingly, existing tools and libraries were used to the maximum extent practicable. The following sections explain how the prototype was implemented; a step-by-step example of how the prototype functions is presented in Section 8.3.

8.2. Implementation

The diagram in Chapter 3 depicting the DRUID methodology, repeated in Figure 12 for convenience, shows a notional architecture. The implementation deviates from this notional depiction, but the same steps are implemented.

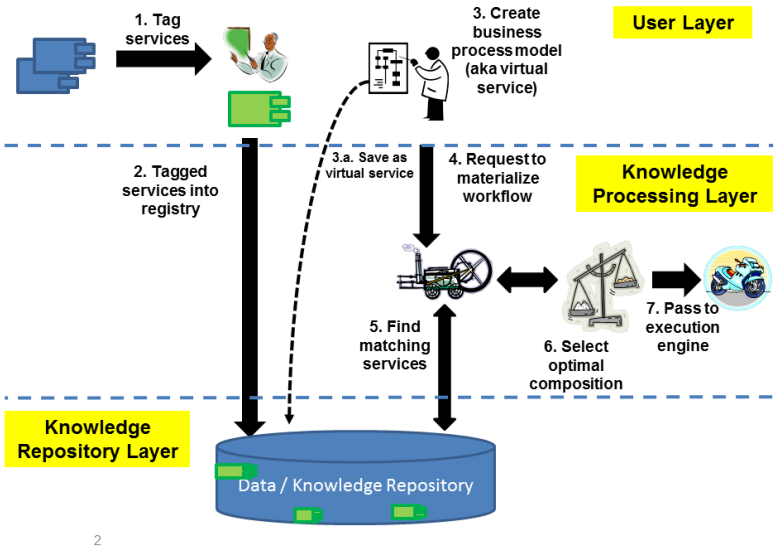

Figure 12: Overview of the DRUID methodology

The prototype implementation consists of four main layers: a service description layer, a model creation layer, a service composition layer, and an optimization layer. Each of these is described in the following subsections:

8.2.1. Service Description Layer

The service description layer consists of a tool for creating service descriptions that conform to the OGMA service description model, together with a registry that stores the service descriptions and makes them searchable.

Because the service descriptions are defined using OWL, a tool that can create OWL-compliant models was preferable to a generic XML editing tool. In order to take advantage of the power of machine reasoning, it is desirable to use a language to which reasoners could be applied. Because there are a number of reasoners that support the OWL language (e.g., HermiT and Pellet), it was also desirable for the selected tool to support a full-featured OWL programming interface. After analysis of the available tools, the Protégé tool developed by Stanford University (62) was selected as it is a mature, well-maintained OWL tool, and it supports the Manchester OWL syntax (63). The Manchester OWL syntax has a mature Application Programming Interface (API).

The Protégé tool has an additional feature, in that it can store OWL instance data as well as OWL models. This makes it possible to use Protégé as both the service description creation tool and the service registry, with service descriptions stored in an OWL file that can be read using the Manchester OWL API.

8.2.2. Model Creation Layer

The model creation layer consists of the tool used to create the process model that will be automated, in addition to a custom BPMN parsing library to read the BPMN model extensions from the process model. In order to maximize standards compliance of the XML representations of the BPMN models, and ensure they could be easily parsed, a modeling tool that is BPMN 2.0 compliant is ideal. (As discussed in Section 2.2.1, BPMN did not have a standard XML representation prior to version 2.0.)

After reviewing the available commercial- and open-source modeling tools, the Sparx Enterprise Architect tool was selected. The SparxEA tool is a full-featured Model-Driven Architecture (MDA) tool that includes support for BPMN modeling and is fully BPMN 2.0 compliant. The ease with which a BPMN model can be exported as XML was another strong factor in its favor. Finally, the cost of the Sparx tool was a significant consideration, as its license cost is an order of magnitude lower than that of any other commercial BPMN tool.

Once the SparxEA BPMN modeling tool was selected, it was necessary to determine how best to insert the BPMN extensions into the BPMN models created by the tool. After assessing the XML generation capabilities of the Sparx tool, it was determined that adding new XML elements would require major modifications to the base tool. These modifications are impractical without access to the tool's source code, and modifying a modeling tool is beyond the scope of this research. After careful analysis, it was determined that the BPMN extensions could be captured in the documentation fields for each model element and readily parsed out of the XML specification generated by the tool.

To parse the model extensions out of the XML specification so they could be used for service composition, a BPMN parsing library was written in Java that takes as input a BPMN model annotated with extensions, reads out those extensions from the documentation fields, and returns as output a series of data objects representing each of the activities in the process model. These data objects have methods for reading out the activity name, the activity's semantic type, and the semantic type of each input and output parameter.

8.2.3. Service Composition Layer

The service composition layer consists of a set of intelligent software agents that receive the process model from the user, invoke the BPMN parser, and find candidate services for each of the activities in the model, based on the semantic descriptions as expressed in the BPMN extensions.

The service composition layer was implemented using an agent-based approach for several reasons. First, the adoption of the JADE Agent Framework, which embodies the the communications protocols published by the Foundation for Independent Physical Agents (FIPA) (64), provides a convenient communications framework with a defined semantics and simplifies communication among multiple parallel processes without the overhead of manually managing a multi-threaded application. Second, an agent-based system makes it possible to use intelligent agents that are empowered to negotiate with each other over selected aspects of the process composition problem, such as the cost to invoke a given service. Third, the semantics of the FIPA communications protocols

makes it easier or agents to reason about the status of the service composition and the process model contents.

There are three main classes of agents used in the prototype: 1) support agents that process the BPMN model and coordinate the search for services; 2) task agents that search the registry for services that match a specific activity in the process model, and 3) service agents, that represent the services to be composed, and serve as their proxies.

Support Agents

The support agents are those agents that perform utility duties such as handing interactions with the user interface, reading the ontology, and doing graph analysis. These agents form the core of the prototype and do most of the computation required to analyze the user's process model and to assess candidate services.

GuiAgent

The GuiAgent's function is to communicate with the graphical user interface (GUI), receiving input from the user and passing that input to the appropriate agent for further processing. The GuiAgent also updates the GUI display based on the progress of the analysis.

ModelReaderAgent

The ModelReaderAgent's purpose is to read the XML output of the BPMN model and extract the semantic information captured in the BPMN modeling extensions described in Chapter 6. The ModelReaderAgent parses the model using the custom BPMN parser described in Section 8.2.1.

GraphAgent

The GraphAgent is responsible for receiving information about candidate services and making an initial assessment about their suitability. The GraphAgent creates and analyzes a directed graph where nodes represent each of the candidate services and edges are asserted wherever two services can be composed together (i.e., the outputs of one service include the required inputs of the other service).

OntologyReaderAgent

The function of the OntologyReaderAgent is to use the Manchester OWL API to read the ontology and search for services meeting specific criteria, such as performing the type of task specified by an activity in the BPMN model. The OntologyReaderAgent passes information about discovered services back to the requesting agent.

Because the OntologyReaderAgent reads information from the service description ontology, it already has all the information necessary to invoke the optimization process described in Section 8.2.4. For this reason, the OntologyReaderAgent is used to invoke the optimization process.

BrokerAgent

The BrokerAgent is a specialized agent whose purpose is to broker negotiations among service agents whenever such negotiation is indicated. The BrokerAgent may request initial bids and updated bids from service agents if the service agents are empowered to alter their QoS metrics, or if the user requests an updated offer.

Selection Agents

The selection agents search for specific service offerings to match an individual activity in the BPMN model. All selection agents are based on a single TaskAgent

template, but each one of them is specialized to search for a specific type of service. Once the BPMN model is parsed to retrieve the semantic information for each activity, the ModelReaderAgent creates a specialized TaskAgent for each semantic description of an activity. As a result, the set of TaskAgents that is initiated is specific to the BPMN model that was submitted.

Service Agents

Service agents are specialized agents that are not necessary to the functioning of the prototype, but can provide enhanced functionality when they are available. A service agent is a FIPA-compliant intelligent agent that is a proxy for one of the services available in the registry. The primary purpose of a service agent is to represent the service to the user and negotiate QoS terms if the user is interested.

8.2.4. Optimization Layer

The optimization layer implements the optimization model described in detail in Section 4. The optimization layer is implemented using OPL and IBM's CPLEX modeling library.

The optimization layer accepts as input a data file created by the OntologyReaderAgent, and a model definition file, and passes those to the CPLEX library. The optimization model verifies that the candidate services can be composed together to materialize the process described in the original BPMN model, and applies other constraints based on user preferences (e.g., minimize cost) that are used to determine which composition of services is optimal. After processing, the CPLEX library returns the optimal service composition based on the QoS parameters supplied in the data

file. An example of the CPLEX output can be found in Appendix B: CPLEX OPL Output.

8.3.Execution Example

What follows is a simple execution of the proof-of-concept prototype, beginning with the creation of service descriptions, proceeding through the creation and annotation of a BPMN model, and culminating in the identification of an optimal service composition based on QoS parameters.

The process begins with creating service descriptions using the Protégé tool, as depicted in Figure 13. The information needed to fully describe the semantics of the service, including its inputs, outputs, and QoS parameters, is entered into the service description model.

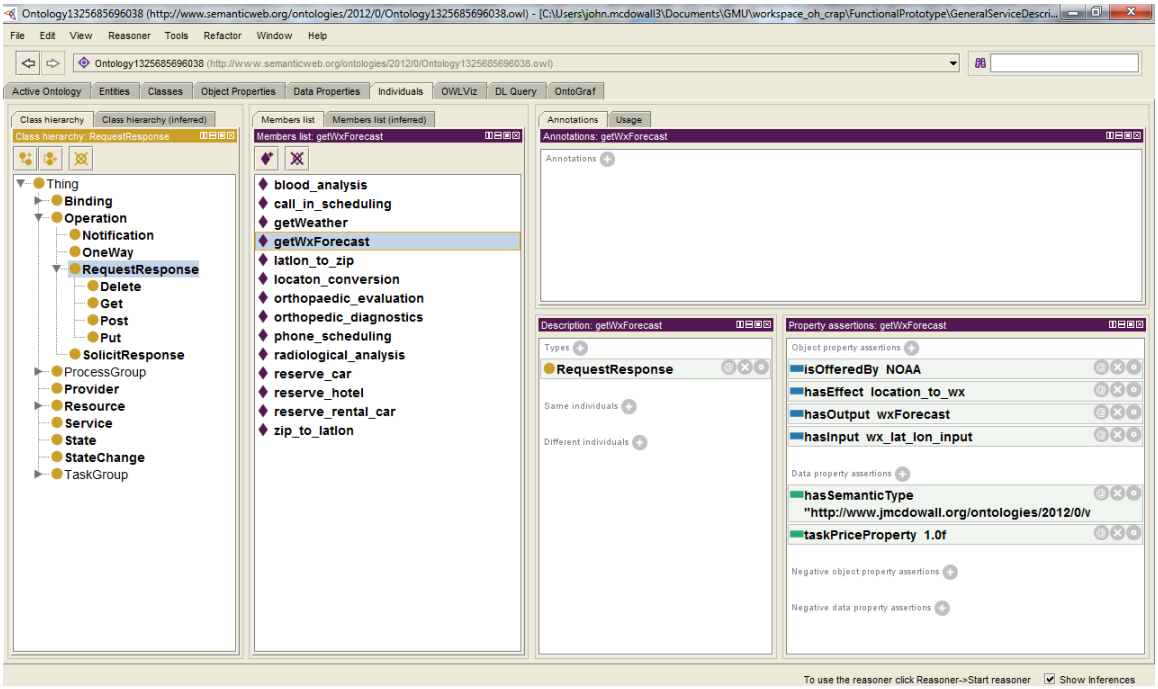

Figure 13: Creating a service description

After each of the available services is fully described using the Protégé tool, there is sufficient information to enable the matching of services to activities and services to services.

The next step in the process is for a business user or analyst to create a BPMN-semantically-extended model using the SparxEA tool. The creation of the model is shown in Figure 14, which shows a standard graphical BPMN model.

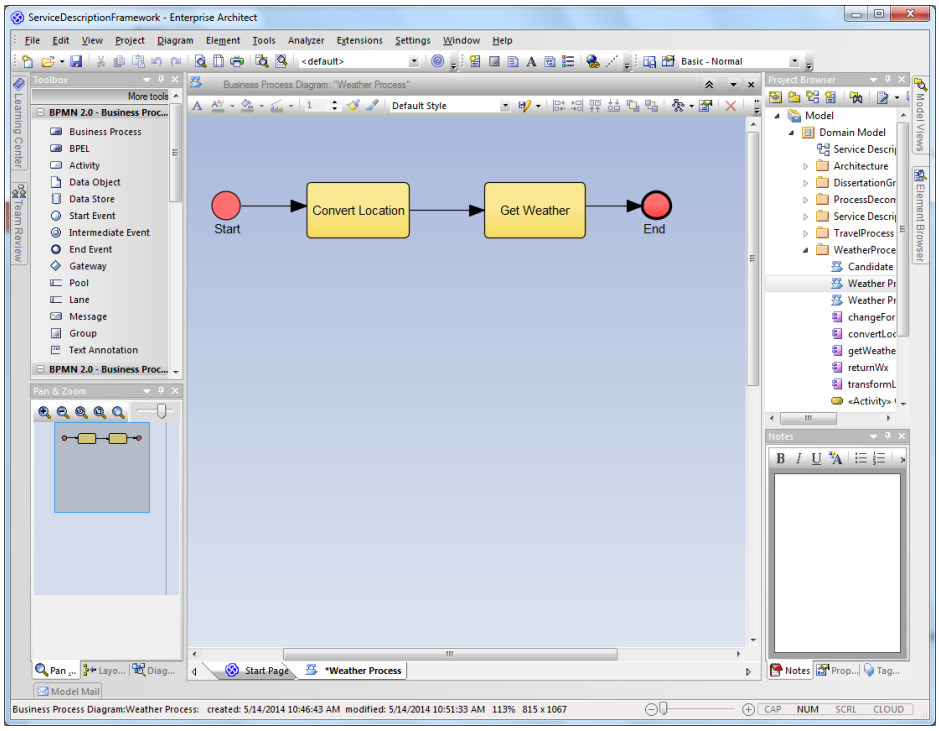

Figure 14: Specification of a BPMN process model

Once the BPMN model has been created, it is necessary to add the semantic annotations defined by the BPMN extensions specified as part of this research. As described earlier, these annotations are captured in the BPMN documentation field as depicted in Figure 15.

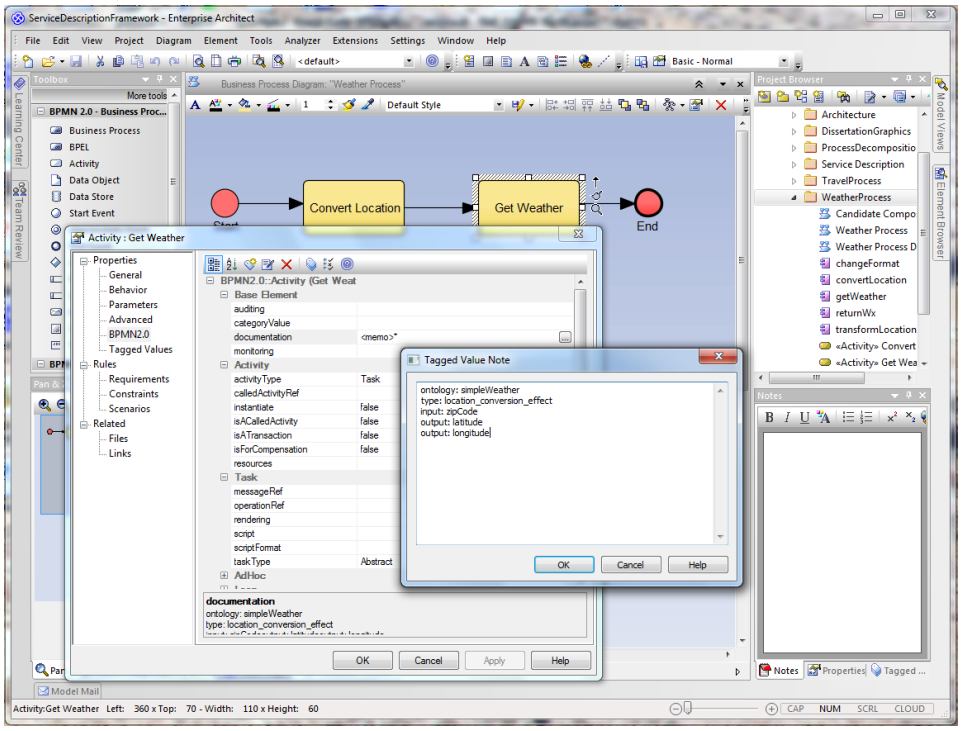

Figure 15: Annotation of semantic information on an activity

Once the semantic annotation of the model is completed, it is exported to XML-format that conforms to the BPMN 2.0 XML specification. The annotated model is now ready for parsing and subsequent analysis by the agents described earlier. Selection of the process model's XML instantiation is depicted in Figure 16.

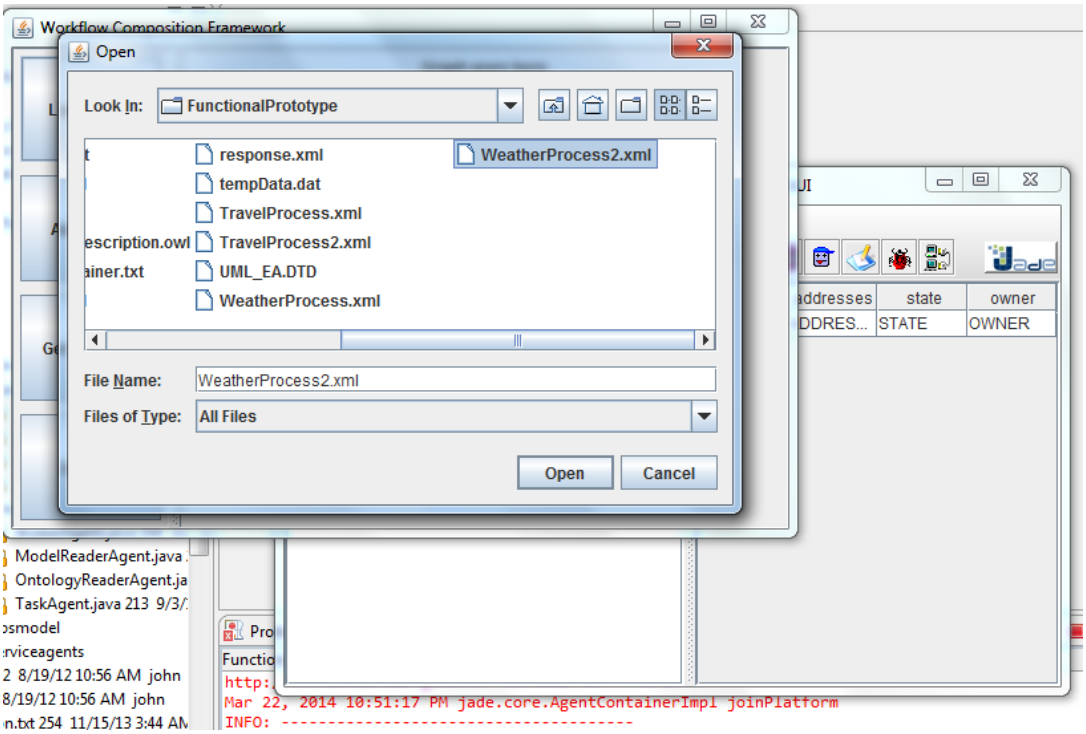

Figure 16: Selecting a process model

Once the process model is selected in the GUI, the model is passed to the ModelReaderAgent. Using the custom BPMN parsing library, the ModelReaderAgent reads the model and parses the semantic information about activity types, inputs, and outputs from the documentation fields in the model.

Figure 17 shows the JADE console with a variety of agents active, including two specialized TaskAgents, named end::GetWeatherForLatLon and start::GetLatLonForZip. Each of these agents is dynamically created by the ModelReaderAgent based on the semantic types of the two activities in the sample model.

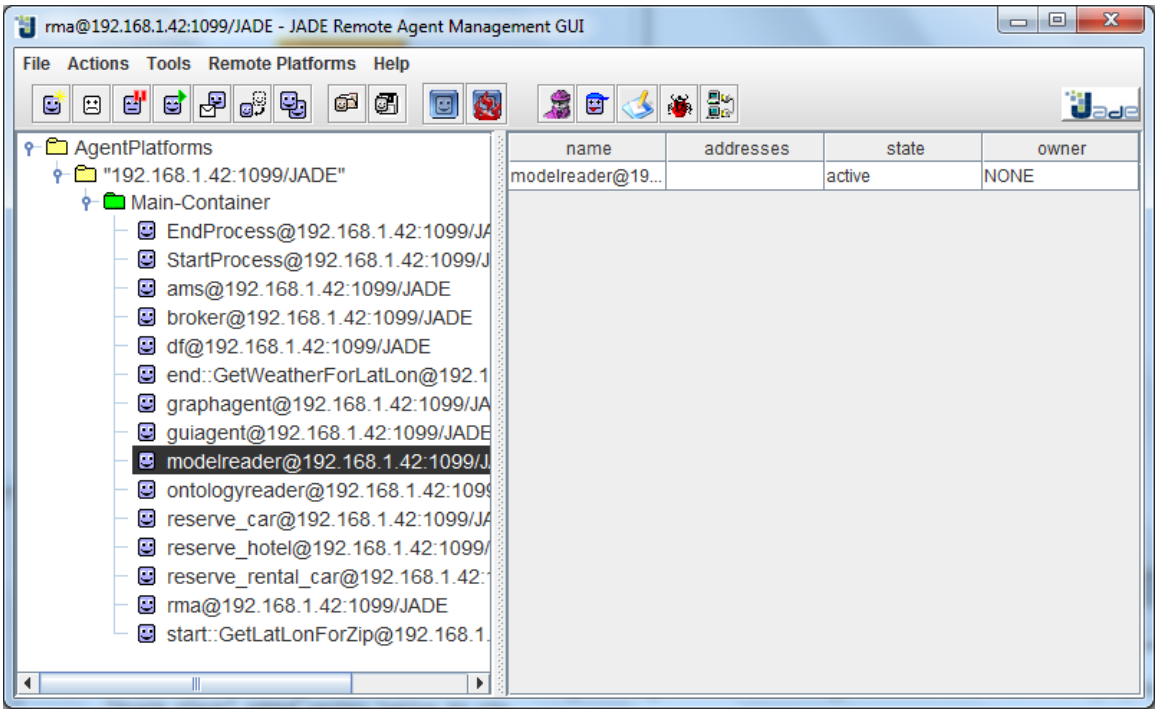

Figure 17: The JADE agent framework during model processing

Each of the TaskAgents independently contacts the OntologyReaderAgent and requests a list of services that perform the type of function that TaskAgent is specialized for. For each request message it receives from a service selection agent, the OntologyReaderAgent queries the service registry to discover which services are available that match the semantic type contained in the request.

The service descriptions for each matching service are read from the registry and information about the task type, inputs, and outputs are passed back to the requesting TaskAgent. Each of these service discovery agents passes this information along to the GraphAgent for further processing.

The GraphAgent's first action is to add each service as a node to a directed graph and pass the resulting graph back to the GuiAgent for display to the user. This initial graph is displayed as shown in Figure 18. This graph display includes two special nodes, named "start" and "end" that are used to denote the beginning and the end of the process.

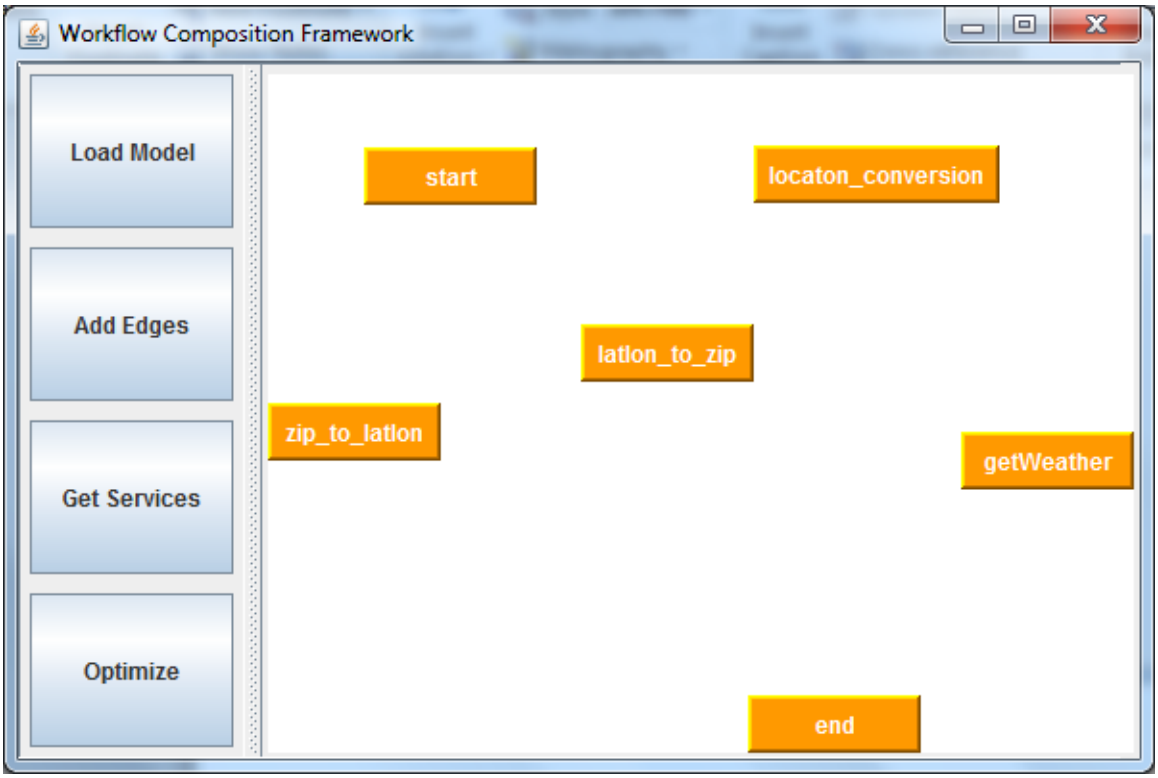

Figure 18: GUI showing services that match process activities

Once the initial graph (with no edges) has been displayed, the user can click the "Add Edges" button to analyze the services in the graph to determine what compositions are possible. The GraphAgent analyzes the inputs and outputs of each node in the graph to determine which services may be composed together and in what order. After

completing this analysis, the GraphAgent adds directed edges to the graph for each potential composition and passes the updated graph to the GuiAgent. The GuiAgent updates the GUI with the completed graph as shown in Figure 19.

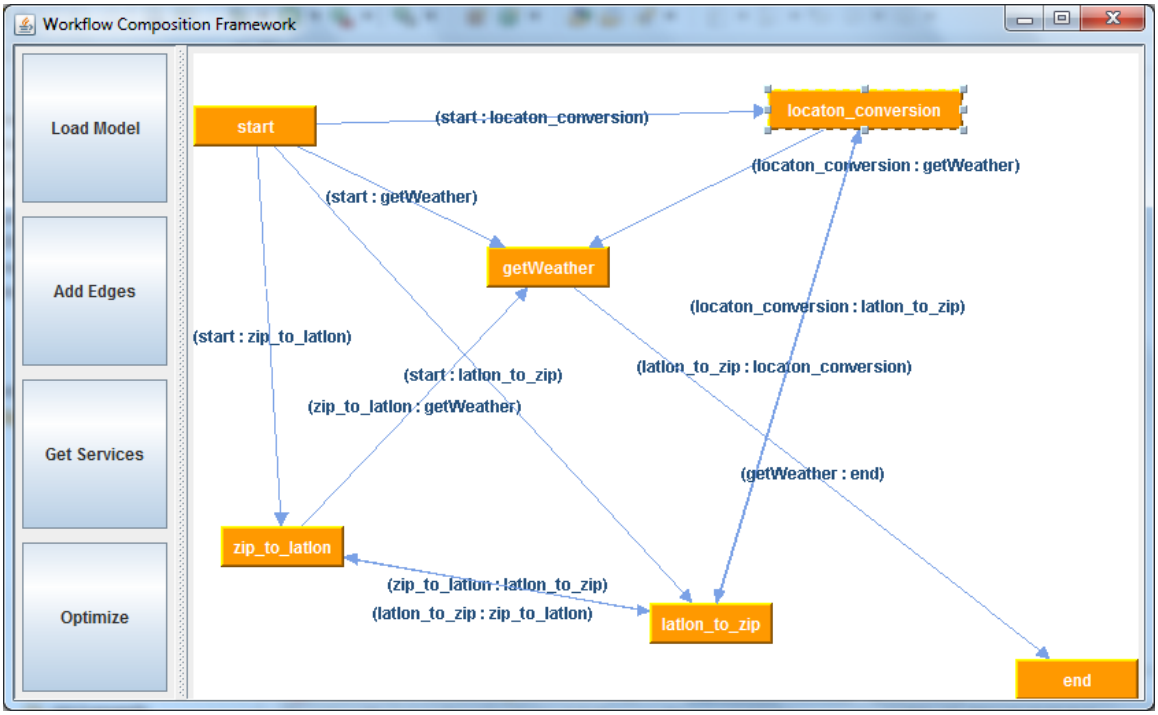

Figure 19: GUI with initial all possible service compositions

An examination of the graph in Figure 19 shows that some edges are bi-directional, as the inputs and outputs of the services represented by those nodes support combining them in two different orders. For example, the zip_to_latlon service and the latlon_to_zip service both have the semantic type “location_conversion.” One converts a zip code to a latitude / longitude coordinate, and the other converts a latitude / longitude

coordinate to a zip code. This behavior can result in cycles in the graph, which would mean cyclic service invocations that never complete the process defined by the model.

In order to eliminate these cycles and find candidate service compositions that do not include extraneous services, the GraphAgent analyzes the graph using the Floyd-Warshall algorithm to find acyclic paths through the graph from the start node to the end node. Each of these paths represents a composition of services that will materialize the process from start to end.

Each of the candidate compositions discovered through graph analysis is then passed to the OntologyReaderAgent to begin the optimization process. The OntologyReaderAgent examines each of the services in each candidate composition and queries the service registry for the QoS attributes of that service. As the QoS attributes for each service are read from the registry, the relevant information about that service as described in Section 4 (task type, input and output semantic parameters, QoS attributes) is written to a data file that will be used by the optimization process.

Once the optimization data file is written, the OntologyReaderAgent invokes the CPLEX engine, passing the data file and a reference to the optimization model and starting the optimization analysis. Upon completion of the analysis, the optimization model returns the recommended service composition based on the user's QoS preference (e.g., minimize cost). (An example of the output from the CPLEX engine can be seen in Appendix B: CPLEX OPL Output.) Continuing with the above example, the two possible compositions are the following:

- Option A: location_conversion → getWeather

- Option B: zip_to_latlon → getWeather

The CPLEX analysis of the QoS parameters of each service, together with the user preference to minimize cost, results in an optimal service composition of location_conversion → getWeather. Given the simple case of this example, it is easy to manually verify the correctness of this assessment. First, the outputs of the location_conversion service are compared to the inputs of the getWeather service, and it is verified that they may be composed together. The same comparison is performed to the outputs of the zip_to_latlon service and the inputs of the getWeather service, also these two may be composed together.

Finally, it can be verified that the QoS assessment is correct by computing the total cost of each composition and verifying that the cost of Option A is less than the cost of Option B. Note that the same verification process is applicable to a more complex comparison, but this verification process becomes more difficult with more services and more QoS parameters.

8.4.Scalability

To assess the scalability of the optimization implementation, it was executed with an array of increasingly complex inputs to compare the time required to complete the optimization calculation. The results of this testing are shown in Figure 20.

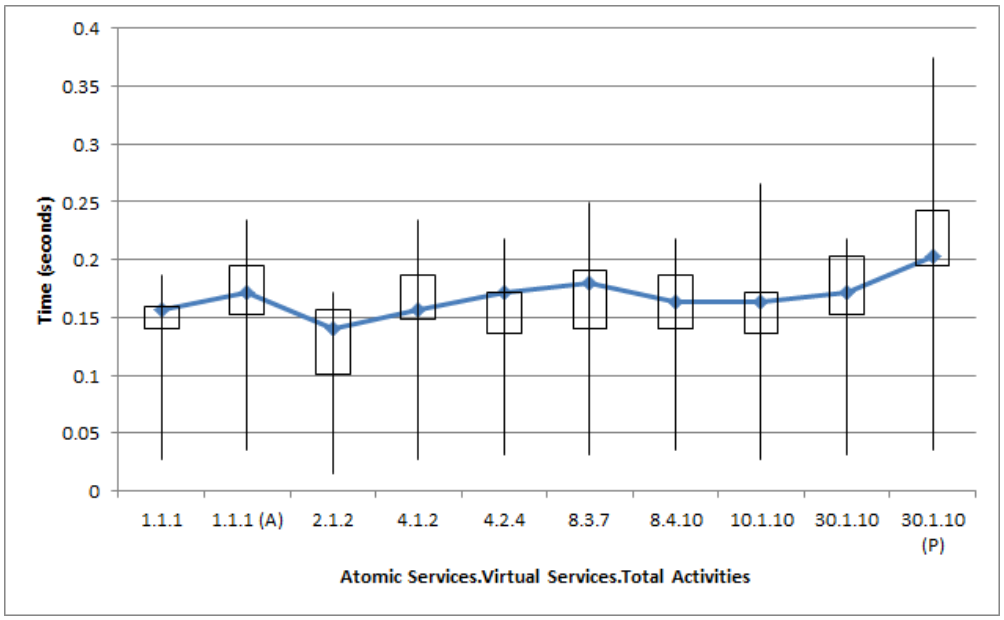

Figure 20: Optimization Scalability

To generate this graph, each test case was executed ten times and the results were plotted using a box-and-whisker diagram that begins with the simplest cases on the left and progresses to the most complex cases on the right. The maximum and minimum run time for each test is shown by the lines, with the boxes representing the 25th through 75th percentiles. The blue line shows the median value for each test run. As shown in the graph, the execution time is relatively stable across most of the test cases but begins to increase with the most complex test cases.

The column labels in the chart represent the complexity of the test case using a dot notation. The first number represents the number of atomic services, the second number represents the number of virtual services including the root service, and the third

number represents the total number of activities in all of the virtual services. A more detailed explanation of the test cases is provided in Table 45.

Table 45: Scalability Test Cases

Column Label	Explanation
1.1.1	One atomic service, one virtual service containing a single activity
1.1.1(A)	One atomic service, one virtual service containing a single activity
2.1.2	Two atomic services, one virtual service containing two activities
4.1.2	Four atomic services, one virtual service containing two activities
4.2.4	Four atomic services, two virtual services containing two activities each
8.3.7	Four atomic services, three virtual services containing a total of seven activities
8.4.10	Eight atomic services, four virtual services containing ten total activities and nested three deep
10.1.10	Ten atomic services, one virtual service containing ten activities
30.1.10	Thirty atomic services, one virtual service containing ten activities
30.1.10 (P)	Thirty atomic services, one virtual service containing ten activities and multiple concurrent activities (i.e., parallel paths)

Given the definitions in Chapter 4, it can be expected that the total nnumber of activities that must be mapped to services (including all virtual services) will be the driving factor in the time required to complete the optimization assessment. As the data

in Figure 20 show, the total number of activities alone is not the determining factor in driving the time required to complete the optimization. Instead, the driver is the number of activities combined with the number of concurrent activities. Greater concurrency intuitively makes the optimization computation more complex and this data validates that intuition.

Even given the increased processing time required to optimize a virtual service composition with multiple concurrent activities, it can be seen that the optimization implementation is scalable.

9. CONCLUSIONS AND FUTURE WORK

This research demonstrates the feasibility of automated service composition and selection based on a workflow model specified using a standards-based Process Modeling Laguage, BPMN. This chapter describes the key contributions of this research, conclusions, and areas for future research.

9.1. Key Contributions

The feasibility of automated service composition has been demonstrated through the key contributions of this research as explained in the following sections.

9.1.1. DRUID Methodology

This research developed the DRUID service composition methodology, a multi-step process for semantically tagging services, developing a semantically-tagged process specification, and automating the composition of the tagged services, then computing an optimal service composition that materializes the process model.

9.1.2. BPMN-S Extension

Another contribution of this research is the BPMN-S extension to the BPMN modeling language, thereby providing a means for tagging activities in a BPMN model with the semantic information necessary to enable the automated matching of services to each process activity.

9.1.3. OGMA Service Description Language

To enable both the activity-to-service matchmaking and the service-to-service matchmaking necessary to compose services, this research develops the OGMA service description language, specified in OWL. This language further enables the comparison of service semantics and the application of basic machine reasoning using tools such as the HermiT reasoner. When applied to the service descriptions,the HermiT reasoner proved useful in three respects. First, it ensured that the OGMA ontology itself was internally consistent. Second, it ensured that every service description was valid. Finally, on several occaisions the reasoner interpreted descriptions of different services as the same service where that equivalence was not intended, thereby highlighting errors in specific service descrpitions.

9.1.4. SUCELLOS Quality of Service Model

To compare service compositions based on QoS metrics, this research develops the SUCELLOS QoS model, an OWL-based ontology based on the QoS model developed by INRIA and described in detail in Chapter 5. The SUCELLOS model captures the QoS metrics of services and enables comparison of QoS metrics among services and across service compositions.

9.1.5. ECNE Optimization Model

This research also develops the ECNE service composition optimization model. The ECNE model includes a formal definition of a virtual service (also known as a business process) as well as formal definitions of an optimal service composition and the QoS metric aggregation used to assess service composition optimality.

9.1.6. Proof-of-Concept Prototype

Finally, this research develops a proof-of-concept prototype that applies the DRUID methodology and other contributions of this research to demonstrate the feasibility of the DRUID methodology as well as the usability of the OGMA, SUCELLOS, and ECNE models.

9.2. Conclusions

This research develops a proof of concept prototype that ties together all of the main components of the DRUID methodology into an operable system that accepts a process model as input and returns an optimal service composition based on the QoS metrics of available services.

The OGMA service description language provides a mechanism for describing services of all types, including web services and physical services, with the semantic detail necessary to enable automated activity-to-service matchmaking and service-to-service matchmaking. As a superset of both the WSDL and WADL service description formats the OGMA language is backward-compatible with existing service description formats, allowing developers to leverage existing service descriptions when creating OGMA -compliant service descriptions. As an OWL-based model, OGMA supports machine reasoning and inference across service descriptions.

The extensions to the BPMN modeling specification, developed in this research, provide a means for appending semantic metadata to activities in a BPMN model. This additional semantic information enables automated activity-to-service matchmaking based on a process modeling language that is in wide use among business analysts and users.

The SUCELLOS QoS model developed in this research provides a robust and flexible QoS model that is suitable for describing services of all types and can be readily extended to include additional QoS parameters, if needed. Because the Sucellos model was developed in OWL, it readily integrates with service descriptions conforming to the OGMA language.

To enable the selection of optimal service compositions, this research develops the ECNE optimization formalism, a formal model that describes processes, services, and QoS parameters in a way that supports the mathematical analysis service compositions to select the optimal composition based on QoS parameters. This optimization model was implemented in OPL using the IBM CPLEX suite to demonstrate the efficacy of the formalism and the optimization process.

9.3.Future Work

The dynamic composition of services within a Service-Oriented Architecture remains a rich field for additional research. Some areas of potential interest for future research are described below.

9.3.1. Service Security

One of the main areas for additional research is to develop interoperable security models that can be used for web services. The range of security models currently in use, from username / password through Public Key Infrastructure (PKI) certificates, makes it difficult to integrate web services that use different security models; this research assumes that security was not a consideration at this time. Within a single enterprise, it is possible to standardize on a single security approach. However, standardizing multiple

enterprises on a single security model is difficult and costly. To be operationally useful, a service composition framework must provide some means of composing services that use different security models. This remains an area of active research and is ripe for further investigation.

9.3.2. Quality of Service

This research explored the application of QoS metrics to optimal service composition selection, but this research only explored a limited number of QoS metrics. The application of additional metrics, such as data throughput or the ability of a service to adapt to intermittent network connectivity issues, provides a fertile area for additional research.

9.3.3. Composition Execution

This research stopped short of attempting to execute the selected service composition. However, converting the optimal service composition into an executable format such as BPEL (reference) offers additional challenges for future researchers. One such challenge is to automate the development of a BPEL process specification that includes error handling robust enough for operational use. Another challenge would to apply the "BPEL for People" (reference) specification to the methodology developed by this research in order to enable the execution of service compositions that include both physical and web services.

APPENDIX A: OWL SPECIFICATION OF OGMA DESCRIPTION LANGUAGE

```
<?xml version="1.0"?>

<!DOCTYPE Ontology [
    <!ENTITY xsd "http://www.w3.org/2001/XMLSchema#" >
    <!ENTITY xml "http://www.w3.org/XML/1998/namespace" >
    <!ENTITY rdfs "http://www.w3.org/2000/01/rdf-schema#" >
    <!ENTITY rdf "http://www.w3.org/1999/02/22-rdf-syntax-ns#" >
]>

<Ontology xmlns="http://www.w3.org/2002/07/owl#"

xml:base="http://www.semanticweb.org/ontologies/2012/0/Ontology1325685696038.owl"
     xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
     xmlns:xsd="http://www.w3.org/2001/XMLSchema#"
     xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
     xmlns:xml="http://www.w3.org/XML/1998/namespace"

ontologyIRI="http://www.semanticweb.org/ontologies/2012/0/Ontology1325685696038.owl">
    <Prefix name="" IRI="http://www.w3.org/2002/07/owl#"/>
    <Prefix name="owl" IRI="http://www.w3.org/2002/07/owl#"/>
    <Prefix name="rdf" IRI="http://www.w3.org/1999/02/22-rdf-syntax-ns#"/>
    <Prefix name="xsd" IRI="http://www.w3.org/2001/XMLSchema#"/>
    <Prefix name="rdfs" IRI="http://www.w3.org/2000/01/rdf-schema#"/>

<Import>http://www.jmcdowall.org/ontologies/QoSOntology.owl</Import>
    <Declaration>
        <Class IRI="#Animate"/>
    </Declaration>
    <Declaration>
        <Class IRI="#Binding"/>
    </Declaration>
    <Declaration>
        <Class IRI="#Delete"/>
    </Declaration>
    <Declaration>
        <Class IRI="#Element"/>
    </Declaration>
    <Declaration>
        <Class IRI="#Get"/>
    </Declaration>
    <Declaration>
```

```
    <Class IRI="#Inanimate"/>
</Declaration>
<Declaration>
    <Class IRI="#Message"/>
</Declaration>
<Declaration>
    <Class IRI="#NetworkBinding"/>
</Declaration>
<Declaration>
    <Class IRI="#Notification"/>
</Declaration>
<Declaration>
    <Class IRI="#OneWay"/>
</Declaration>
<Declaration>
    <Class IRI="#Operation"/>
</Declaration>
<Declaration>
    <Class IRI="#PhysicalBinding"/>
</Declaration>
<Declaration>
    <Class IRI="#PhysicalResource"/>
</Declaration>
<Declaration>
    <Class IRI="#Post"/>
</Declaration>
<Declaration>
    <Class IRI="#Provider"/>
</Declaration>
<Declaration>
    <Class IRI="#Put"/>
</Declaration>
<Declaration>
    <Class IRI="#RequestResponse"/>
</Declaration>
<Declaration>
    <Class IRI="#Resource"/>
</Declaration>
<Declaration>
    <Class IRI="#Service"/>
</Declaration>
<Declaration>
    <Class IRI="#SolicitResponse"/>
</Declaration>
<Declaration>
    <Class IRI="#State"/>
</Declaration>
<Declaration>
    <Class IRI="#StateChange"/>
</Declaration>
<Declaration>
    <Class IRI="#VirtualResource"/>
</Declaration>
<Declaration>
```

```
    <ObjectProperty IRI="#hasBinding"/>
</Declaration>
<Declaration>
    <ObjectProperty IRI="#hasCause"/>
</Declaration>
<Declaration>
    <ObjectProperty IRI="#hasCurrentState"/>
</Declaration>
<Declaration>
    <ObjectProperty IRI="#hasEffect"/>
</Declaration>
<Declaration>
    <ObjectProperty IRI="#hasElement"/>
</Declaration>
<Declaration>
    <ObjectProperty IRI="#hasFinalState"/>
</Declaration>
<Declaration>
    <ObjectProperty IRI="#hasInitialState"/>
</Declaration>
<Declaration>
    <ObjectProperty IRI="#hasInput"/>
</Declaration>
<Declaration>
    <ObjectProperty IRI="#hasOperation"/>
</Declaration>
<Declaration>
    <ObjectProperty IRI="#hasOutput"/>
</Declaration>
<Declaration>
    <ObjectProperty IRI="#hasPrecondition"/>
</Declaration>
<Declaration>
    <ObjectProperty IRI="#hasRecommendedPrecondition"/>
</Declaration>
<Declaration>
    <ObjectProperty IRI="#hasRequiredPrecondition"/>
</Declaration>
<Declaration>
    <ObjectProperty IRI="#hasState"/>
</Declaration>
<Declaration>
    <ObjectProperty IRI="#isElementOf"/>
</Declaration>
<Declaration>
    <ObjectProperty IRI="#isInputOf"/>
</Declaration>
<Declaration>
    <ObjectProperty IRI="#isOfferedBy"/>
</Declaration>
<Declaration>
    <ObjectProperty IRI="#offers"/>
</Declaration>
<Declaration>
```

```
    <DataProperty IRI="#hasCity"/>
</Declaration>
<Declaration>
    <DataProperty IRI="#hasCommunicationProtocol"/>
</Declaration>
<Declaration>
    <DataProperty IRI="#hasContentSubType"/>
</Declaration>
<Declaration>
    <DataProperty IRI="#hasContentType"/>
</Declaration>
<Declaration>
    <DataProperty IRI="#hasCountry"/>
</Declaration>
<Declaration>
    <DataProperty IRI="#hasDataType"/>
</Declaration>
<Declaration>
    <DataProperty IRI="#hasDescription"/>
</Declaration>
<Declaration>
    <DataProperty IRI="#hasFacebookIdentifier"/>
</Declaration>
<Declaration>
    <DataProperty IRI="#hasGenus"/>
</Declaration>
<Declaration>
    <DataProperty IRI="#hasIdentifier"/>
</Declaration>
<Declaration>
    <DataProperty IRI="#hasMaxCardinality"/>
</Declaration>
<Declaration>
    <DataProperty IRI="#hasMinCardinality"/>
</Declaration>
<Declaration>
    <DataProperty IRI="#hasNAICSCode"/>
</Declaration>
<Declaration>
    <DataProperty IRI="#hasName"/>
</Declaration>
<Declaration>
    <DataProperty IRI="#hasPhoneNumber"/>
</Declaration>
<Declaration>
    <DataProperty IRI="#hasPointOfContact"/>
</Declaration>
<Declaration>
    <DataProperty IRI="#hasPostalCode"/>
</Declaration>
<Declaration>
    <DataProperty IRI="#hasSemanticType"/>
</Declaration>
<Declaration>
```

```
    <DataProperty IRI="#hasSoapStyle"/>
</Declaration>
<Declaration>
    <DataProperty IRI="#hasSoapTransport"/>
</Declaration>
<Declaration>
    <DataProperty IRI="#hasSpecies"/>
</Declaration>
<Declaration>
    <DataProperty IRI="#hasStateOrProvince"/>
</Declaration>
<Declaration>
    <DataProperty IRI="#hasStatus"/>
</Declaration>
<Declaration>
    <DataProperty IRI="#hasStreetAddress"/>
</Declaration>
<Declaration>
    <DataProperty IRI="#hasURI"/>
</Declaration>
<Declaration>
    <DataProperty IRI="#hasURL"/>
</Declaration>
<Declaration>
    <DataProperty IRI="#hasYelpIdentifier"/>
</Declaration>
<Declaration>
    <DataProperty IRI="#isDescribedByOntology"/>
</Declaration>
<Declaration>
    <NamedIndividual IRI="#x_ray"/>
</Declaration>
<SubClassOf>
    <Class IRI="#Animate"/>
    <Class IRI="#PhysicalResource"/>
</SubClassOf>
<SubClassOf>
    <Class IRI="#Animate"/>
    <DataExactCardinality cardinality="1">
        <DataProperty IRI="#hasGenus"/>
        <Datatype abbreviatedIRI="xsd:string"/>
    </DataExactCardinality>
</SubClassOf>
<SubClassOf>
    <Class IRI="#Animate"/>
    <DataExactCardinality cardinality="1">
        <DataProperty IRI="#hasSpecies"/>
        <Datatype abbreviatedIRI="xsd:string"/>
    </DataExactCardinality>
</SubClassOf>
<SubClassOf>
    <Class IRI="#Delete"/>
    <Class IRI="#RequestResponse"/>
</SubClassOf>
```

```
<SubClassOf>
    <Class IRI="#Element"/>
    <Class IRI="#VirtualResource"/>
</SubClassOf>
<SubClassOf>
    <Class IRI="#Element"/>
    <DataSomeValuesFrom>
        <DataProperty IRI="#hasSemanticType"/>
        <Datatype abbreviatedIRI="xsd:string"/>
    </DataSomeValuesFrom>
</SubClassOf>
<SubClassOf>
    <Class IRI="#Element"/>
    <DataExactCardinality cardinality="1">
        <DataProperty IRI="#hasDataType"/>
        <Datatype abbreviatedIRI="xsd:string"/>
    </DataExactCardinality>
</SubClassOf>
<SubClassOf>
    <Class IRI="#Element"/>
    <DataExactCardinality cardinality="1">
        <DataProperty IRI="#hasMaxCardinality"/>
        <Datatype abbreviatedIRI="xsd:integer"/>
    </DataExactCardinality>
</SubClassOf>
<SubClassOf>
    <Class IRI="#Element"/>
    <DataExactCardinality cardinality="1">
        <DataProperty IRI="#hasMinCardinality"/>
        <Datatype abbreviatedIRI="xsd:integer"/>
    </DataExactCardinality>
</SubClassOf>
<SubClassOf>
    <Class IRI="#Element"/>
    <DataExactCardinality cardinality="1">
        <DataProperty IRI="#hasName"/>
    </DataExactCardinality>
</SubClassOf>
<SubClassOf>
    <Class IRI="#Get"/>
    <Class IRI="#RequestResponse"/>
</SubClassOf>
<SubClassOf>
    <Class IRI="#Inanimate"/>
    <Class IRI="#PhysicalResource"/>
</SubClassOf>
<SubClassOf>
    <Class IRI="#Inanimate"/>
    <DataSomeValuesFrom>
        <DataProperty IRI="#hasDescription"/>
        <Datatype abbreviatedIRI="xsd:string"/>
    </DataSomeValuesFrom>
</SubClassOf>
<SubClassOf>
```

```
        <Class IRI="#Message"/>
        <Class IRI="#VirtualResource"/>
    </SubClassOf>
    <SubClassOf>
        <Class IRI="#Message"/>
        <ObjectSomeValuesFrom>
            <ObjectProperty IRI="#hasElement"/>
            <Class IRI="#Element"/>
        </ObjectSomeValuesFrom>
    </SubClassOf>
    <SubClassOf>
        <Class IRI="#Message"/>
        <ObjectAllValuesFrom>
            <ObjectProperty IRI="#hasElement"/>
            <Class IRI="#Element"/>
        </ObjectAllValuesFrom>
    </SubClassOf>
    <SubClassOf>
        <Class IRI="#NetworkBinding"/>
        <Class IRI="#Binding"/>
    </SubClassOf>
    <SubClassOf>
        <Class IRI="#NetworkBinding"/>
        <DataSomeValuesFrom>
            <DataProperty IRI="#hasCommunicationProtocol"/>
            <DataOneOf>
                <Literal datatypeIRI="&rdf;PlainLiteral">http</Literal>
                <Literal
datatypeIRI="&rdf;PlainLiteral">https</Literal>
                <Literal datatypeIRI="&rdf;PlainLiteral">sms</Literal>
                <Literal datatypeIRI="&rdf;PlainLiteral">smtp</Literal>
            </DataOneOf>
        </DataSomeValuesFrom>
    </SubClassOf>
    <SubClassOf>
        <Class IRI="#NetworkBinding"/>
        <DataSomeValuesFrom>
            <DataProperty IRI="#hasURL"/>
            <Datatype abbreviatedIRI="xsd:string"/>
        </DataSomeValuesFrom>
    </SubClassOf>
    <SubClassOf>
        <Class IRI="#NetworkBinding"/>
        <DataMaxCardinality cardinality="1">
            <DataProperty IRI="#hasSoapStyle"/>
            <DataOneOf>
                <Literal
datatypeIRI="&rdf;PlainLiteral">document</Literal>
                <Literal datatypeIRI="&rdf;PlainLiteral">rpc</Literal>
            </DataOneOf>
        </DataMaxCardinality>
    </SubClassOf>
    <SubClassOf>
        <Class IRI="#NetworkBinding"/>
```

```
    <DataMaxCardinality cardinality="1">
        <DataProperty IRI="#hasSoapTransport"/>
        <Datatype abbreviatedIRI="xsd:string"/>
    </DataMaxCardinality>
</SubClassOf>
<SubClassOf>
    <Class IRI="#Notification"/>
    <Class IRI="#Operation"/>
</SubClassOf>
<SubClassOf>
    <Class IRI="#OneWay"/>
    <Class IRI="#Operation"/>
</SubClassOf>
<SubClassOf>
    <Class IRI="#Operation"/>
    <ObjectSomeValuesFrom>
        <ObjectProperty IRI="#hasBinding"/>
        <Class IRI="#Binding"/>
    </ObjectSomeValuesFrom>
</SubClassOf>
<SubClassOf>
    <Class IRI="#Operation"/>
    <ObjectSomeValuesFrom>
        <ObjectProperty IRI="#hasEffect"/>
        <Class IRI="#StateChange"/>
    </ObjectSomeValuesFrom>
</SubClassOf>
<SubClassOf>
    <Class IRI="#Operation"/>
    <ObjectSomeValuesFrom>
        <ObjectProperty IRI="#hasInput"/>
        <Class IRI="#Resource"/>
    </ObjectSomeValuesFrom>
</SubClassOf>
<SubClassOf>
    <Class IRI="#Operation"/>
    <ObjectSomeValuesFrom>
        <ObjectProperty IRI="#hasOutput"/>
        <Class IRI="#Resource"/>
    </ObjectSomeValuesFrom>
</SubClassOf>
<SubClassOf>
    <Class IRI="#Operation"/>
    <ObjectSomeValuesFrom>
        <ObjectProperty IRI="#hasPrecondition"/>
        <Class IRI="#State"/>
    </ObjectSomeValuesFrom>
</SubClassOf>
<SubClassOf>
    <Class IRI="#Operation"/>
    <DataSomeValuesFrom>
        <DataProperty IRI="#hasSemanticType"/>
        <Datatype abbreviatedIRI="xsd:string"/>
    </DataSomeValuesFrom>
```

```
    </SubClassOf>
    <SubClassOf>
        <Class IRI="#PhysicalBinding"/>
        <Class IRI="#Binding"/>
    </SubClassOf>
    <SubClassOf>
        <Class IRI="#PhysicalBinding"/>
        <DataSomeValuesFrom>
            <DataProperty IRI="#hasCity"/>
            <Datatype abbreviatedIRI="xsd:string"/>
        </DataSomeValuesFrom>
    </SubClassOf>
    <SubClassOf>
        <Class IRI="#PhysicalBinding"/>
        <DataSomeValuesFrom>
            <DataProperty IRI="#hasCommunicationProtocol"/>
            <DataOneOf>
                <Literal
datatypeIRI="&rdf;PlainLiteral">person</Literal>
                <Literal
datatypeIRI="&rdf;PlainLiteral">postal</Literal>
                <Literal
datatypeIRI="&rdf;PlainLiteral">telephone</Literal>
            </DataOneOf>
        </DataSomeValuesFrom>
    </SubClassOf>
    <SubClassOf>
        <Class IRI="#PhysicalBinding"/>
        <DataSomeValuesFrom>
            <DataProperty IRI="#hasCountry"/>
            <Datatype abbreviatedIRI="xsd:string"/>
        </DataSomeValuesFrom>
    </SubClassOf>
    <SubClassOf>
        <Class IRI="#PhysicalBinding"/>
        <DataSomeValuesFrom>
            <DataProperty IRI="#hasPhoneNumber"/>
            <Datatype abbreviatedIRI="xsd:string"/>
        </DataSomeValuesFrom>
    </SubClassOf>
    <SubClassOf>
        <Class IRI="#PhysicalBinding"/>
        <DataSomeValuesFrom>
            <DataProperty IRI="#hasPointOfContact"/>
            <Datatype abbreviatedIRI="xsd:string"/>
        </DataSomeValuesFrom>
    </SubClassOf>
    <SubClassOf>
        <Class IRI="#PhysicalBinding"/>
        <DataSomeValuesFrom>
            <DataProperty IRI="#hasPostalCode"/>
            <Datatype abbreviatedIRI="xsd:string"/>
        </DataSomeValuesFrom>
    </SubClassOf>
```

```
<SubClassOf>
    <Class IRI="#PhysicalBinding"/>
    <DataSomeValuesFrom>
        <DataProperty IRI="#hasStateOrProvince"/>
        <Datatype abbreviatedIRI="xsd:string"/>
    </DataSomeValuesFrom>
</SubClassOf>
<SubClassOf>
    <Class IRI="#PhysicalBinding"/>
    <DataSomeValuesFrom>
        <DataProperty IRI="#hasStreetAddress"/>
        <Datatype abbreviatedIRI="xsd:string"/>
    </DataSomeValuesFrom>
</SubClassOf>
<SubClassOf>
    <Class IRI="#PhysicalResource"/>
    <Class IRI="#Resource"/>
</SubClassOf>
<SubClassOf>
    <Class IRI="#Post"/>
    <Class IRI="#RequestResponse"/>
</SubClassOf>
<SubClassOf>
    <Class IRI="#Provider"/>
    <DataSomeValuesFrom>
        <DataProperty IRI="#hasCity"/>
        <Datatype abbreviatedIRI="xsd:string"/>
    </DataSomeValuesFrom>
</SubClassOf>
<SubClassOf>
    <Class IRI="#Provider"/>
    <DataSomeValuesFrom>
        <DataProperty IRI="#hasCountry"/>
        <Datatype abbreviatedIRI="xsd:string"/>
    </DataSomeValuesFrom>
</SubClassOf>
<SubClassOf>
    <Class IRI="#Provider"/>
    <DataSomeValuesFrom>
        <DataProperty IRI="#hasIdentifier"/>
        <Datatype abbreviatedIRI="xsd:string"/>
    </DataSomeValuesFrom>
</SubClassOf>
<SubClassOf>
    <Class IRI="#Provider"/>
    <DataSomeValuesFrom>
        <DataProperty IRI="#hasName"/>
        <Datatype abbreviatedIRI="rdfs:Literal"/>
    </DataSomeValuesFrom>
</SubClassOf>
<SubClassOf>
    <Class IRI="#Provider"/>
    <DataSomeValuesFrom>
        <DataProperty IRI="#hasPhoneNumber"/>
```

```
        <Datatype abbreviatedIRI="xsd:string"/>
    </DataSomeValuesFrom>
</SubClassOf>
<SubClassOf>
    <Class IRI="#Provider"/>
    <DataSomeValuesFrom>
        <DataProperty IRI="#hasPostalCode"/>
        <Datatype abbreviatedIRI="xsd:string"/>
    </DataSomeValuesFrom>
</SubClassOf>
<SubClassOf>
    <Class IRI="#Provider"/>
    <DataSomeValuesFrom>
        <DataProperty IRI="#hasStateOrProvince"/>
        <Datatype abbreviatedIRI="xsd:string"/>
    </DataSomeValuesFrom>
</SubClassOf>
<SubClassOf>
    <Class IRI="#Provider"/>
    <DataSomeValuesFrom>
        <DataProperty IRI="#hasStreetAddress"/>
        <Datatype abbreviatedIRI="xsd:string"/>
    </DataSomeValuesFrom>
</SubClassOf>
<SubClassOf>
    <Class IRI="#Provider"/>
    <DataSomeValuesFrom>
        <DataProperty IRI="#hasURL"/>
        <Datatype abbreviatedIRI="xsd:string"/>
    </DataSomeValuesFrom>
</SubClassOf>
<SubClassOf>
    <Class IRI="#Put"/>
    <Class IRI="#RequestResponse"/>
</SubClassOf>
<SubClassOf>
    <Class IRI="#RequestResponse"/>
    <Class IRI="#Operation"/>
</SubClassOf>
<SubClassOf>
    <Class IRI="#Resource"/>
    <Class abbreviatedIRI="owl:Thing"/>
</SubClassOf>
<SubClassOf>
    <Class IRI="#Resource"/>
    <ObjectSomeValuesFrom>
        <ObjectProperty IRI="#hasCurrentState"/>
        <Class IRI="#State"/>
    </ObjectSomeValuesFrom>
</SubClassOf>
<SubClassOf>
    <Class IRI="#Resource"/>
    <DataSomeValuesFrom>
        <DataProperty IRI="#hasName"/>
```

```
        <Datatype abbreviatedIRI="xsd:string"/>
    </DataSomeValuesFrom>
</SubClassOf>
<SubClassOf>
    <Class IRI="#Resource"/>
    <DataSomeValuesFrom>
        <DataProperty IRI="#isDescribedByOntology"/>
        <Datatype abbreviatedIRI="xsd:anyURI"/>
    </DataSomeValuesFrom>
</SubClassOf>
<SubClassOf>
    <Class IRI="#Service"/>
    <ObjectSomeValuesFrom>
        <ObjectProperty IRI="#hasOperation"/>
        <Class IRI="#Operation"/>
    </ObjectSomeValuesFrom>
</SubClassOf>
<SubClassOf>
    <Class IRI="#Service"/>
    <DataSomeValuesFrom>
        <DataProperty IRI="#hasNAICSCode"/>
        <Datatype abbreviatedIRI="xsd:integer"/>
    </DataSomeValuesFrom>
</SubClassOf>
<SubClassOf>
    <Class IRI="#Service"/>
    <DataExactCardinality cardinality="1">
        <DataProperty IRI="#hasName"/>
        <Datatype abbreviatedIRI="xsd:string"/>
    </DataExactCardinality>
</SubClassOf>
<SubClassOf>
    <Class IRI="#SolicitResponse"/>
    <Class IRI="#Operation"/>
</SubClassOf>
<SubClassOf>
    <Class IRI="#State"/>
    <DataSomeValuesFrom>
        <DataProperty IRI="#hasDescription"/>
        <Datatype abbreviatedIRI="xsd:string"/>
    </DataSomeValuesFrom>
</SubClassOf>
<SubClassOf>
    <Class IRI="#State"/>
    <DataExactCardinality cardinality="1">
        <DataProperty IRI="#hasStatus"/>
        <Datatype abbreviatedIRI="xsd:string"/>
    </DataExactCardinality>
</SubClassOf>
<SubClassOf>
    <Class IRI="#State"/>
    <DataExactCardinality cardinality="1">
        <DataProperty IRI="#isDescribedByOntology"/>
        <Datatype abbreviatedIRI="xsd:anyURI"/>
```

```
    </DataExactCardinality>
</SubClassOf>
<SubClassOf>
    <Class IRI="#StateChange"/>
    <ObjectSomeValuesFrom>
        <ObjectProperty IRI="#hasFinalState"/>
        <Class IRI="#State"/>
    </ObjectSomeValuesFrom>
</SubClassOf>
<SubClassOf>
    <Class IRI="#StateChange"/>
    <ObjectSomeValuesFrom>
        <ObjectProperty IRI="#hasInitialState"/>
        <Class IRI="#State"/>
    </ObjectSomeValuesFrom>
</SubClassOf>
<SubClassOf>
    <Class IRI="#VirtualResource"/>
    <Class IRI="#Resource"/>
</SubClassOf>
<SubClassOf>
    <Class IRI="#VirtualResource"/>
    <DataMaxCardinality cardinality="1">
        <DataProperty IRI="#hasContentSubType"/>
        <Datatype abbreviatedIRI="xsd:string"/>
    </DataMaxCardinality>
</SubClassOf>
<SubClassOf>
    <Class IRI="#VirtualResource"/>
    <DataMaxCardinality cardinality="1">
        <DataProperty IRI="#hasContentType"/>
        <Datatype abbreviatedIRI="xsd:string"/>
    </DataMaxCardinality>
</SubClassOf>
<DisjointClasses>
    <Class IRI="#Animate"/>
    <Class IRI="#Inanimate"/>
</DisjointClasses>
<DisjointClasses>
    <Class IRI="#Binding"/>
    <Class IRI="#Operation"/>
    <Class IRI="#Resource"/>
</DisjointClasses>
<DisjointClasses>
    <Class IRI="#Delete"/>
    <Class IRI="#Get"/>
    <Class IRI="#Post"/>
    <Class IRI="#Put"/>
</DisjointClasses>
<DisjointClasses>
    <Class IRI="#Element"/>
    <Class IRI="#Message"/>
</DisjointClasses>
<DisjointClasses>
```

```
    <Class IRI="#Notification"/>
    <Class IRI="#OneWay"/>
    <Class IRI="#RequestResponse"/>
    <Class IRI="#SolicitResponse"/>
</DisjointClasses>
<DisjointClasses>
    <Class IRI="#PhysicalResource"/>
    <Class IRI="#VirtualResource"/>
</DisjointClasses>
<ClassAssertion>
    <Class IRI="#Inanimate"/>
    <NamedIndividual IRI="#x_ray"/>
</ClassAssertion>
<SubObjectPropertyOf>
    <ObjectProperty IRI="#hasCurrentState"/>
    <ObjectProperty IRI="#hasState"/>
</SubObjectPropertyOf>
<SubObjectPropertyOf>
    <ObjectProperty IRI="#hasFinalState"/>
    <ObjectProperty IRI="#hasState"/>
</SubObjectPropertyOf>
<SubObjectPropertyOf>
    <ObjectProperty IRI="#hasInitialState"/>
    <ObjectProperty IRI="#hasState"/>
</SubObjectPropertyOf>
<SubObjectPropertyOf>
    <ObjectProperty IRI="#hasRecommendedPrecondition"/>
    <ObjectProperty IRI="#hasPrecondition"/>
</SubObjectPropertyOf>
<SubObjectPropertyOf>
    <ObjectProperty IRI="#hasRequiredPrecondition"/>
    <ObjectProperty IRI="#hasPrecondition"/>
</SubObjectPropertyOf>
<SubObjectPropertyOf>
    <ObjectProperty IRI="#offers"/>
    <ObjectProperty abbreviatedIRI="owl:topObjectProperty"/>
</SubObjectPropertyOf>
<InverseObjectProperties>
    <ObjectProperty IRI="#hasCause"/>
    <ObjectProperty IRI="#hasEffect"/>
</InverseObjectProperties>
<InverseObjectProperties>
    <ObjectProperty IRI="#isElementOf"/>
    <ObjectProperty IRI="#hasElement"/>
</InverseObjectProperties>
<InverseObjectProperties>
    <ObjectProperty IRI="#hasInput"/>
    <ObjectProperty IRI="#isInputOf"/>
</InverseObjectProperties>
<InverseObjectProperties>
    <ObjectProperty IRI="#offers"/>
    <ObjectProperty IRI="#isOfferedBy"/>
</InverseObjectProperties>
<ObjectPropertyRange>
```

```
    <ObjectProperty IRI="#hasBinding"/>
    <Class IRI="#Binding"/>
</ObjectPropertyRange>
<ObjectPropertyRange>
    <ObjectProperty IRI="#hasCause"/>
    <Class IRI="#Operation"/>
</ObjectPropertyRange>
<ObjectPropertyRange>
    <ObjectProperty IRI="#hasCurrentState"/>
    <Class IRI="#State"/>
</ObjectPropertyRange>
<ObjectPropertyRange>
    <ObjectProperty IRI="#hasEffect"/>
    <Class IRI="#StateChange"/>
</ObjectPropertyRange>
<ObjectPropertyRange>
    <ObjectProperty IRI="#hasElement"/>
    <Class IRI="#Element"/>
</ObjectPropertyRange>
<ObjectPropertyRange>
    <ObjectProperty IRI="#hasFinalState"/>
    <Class IRI="#State"/>
</ObjectPropertyRange>
<ObjectPropertyRange>
    <ObjectProperty IRI="#hasInitialState"/>
    <Class IRI="#State"/>
</ObjectPropertyRange>
<ObjectPropertyRange>
    <ObjectProperty IRI="#hasOperation"/>
    <Class IRI="#Operation"/>
</ObjectPropertyRange>
<ObjectPropertyRange>
    <ObjectProperty IRI="#hasPrecondition"/>
    <Class IRI="#State"/>
</ObjectPropertyRange>
<ObjectPropertyRange>
    <ObjectProperty IRI="#isOfferedBy"/>
    <Class IRI="#Provider"/>
</ObjectPropertyRange>
<ObjectPropertyRange>
    <ObjectProperty IRI="#offers"/>
    <Class IRI="#Service"/>
</ObjectPropertyRange>
<SubDataPropertyOf>
    <DataProperty IRI="#hasFacebookIdentifier"/>
    <DataProperty IRI="#hasIdentifier"/>
</SubDataPropertyOf>
<SubDataPropertyOf>
    <DataProperty IRI="#hasStatus"/>
    <DataProperty abbreviatedIRI="owl:topDataProperty"/>
</SubDataPropertyOf>
<SubDataPropertyOf>
    <DataProperty IRI="#hasYelpIdentifier"/>
    <DataProperty IRI="#hasIdentifier"/>
```

```
</SubDataPropertyOf>
<DataPropertyDomain>
    <DataProperty IRI="#hasSpecies"/>
    <Class IRI="#Animate"/>
</DataPropertyDomain>
<DataPropertyRange>
    <DataProperty IRI="#hasCity"/>
    <Datatype abbreviatedIRI="xsd:string"/>
</DataPropertyRange>
<DataPropertyRange>
    <DataProperty IRI="#hasCommunicationProtocol"/>
    <Datatype abbreviatedIRI="xsd:string"/>
</DataPropertyRange>
<DataPropertyRange>
    <DataProperty IRI="#hasContentSubType"/>
    <Datatype abbreviatedIRI="xsd:string"/>
</DataPropertyRange>
<DataPropertyRange>
    <DataProperty IRI="#hasContentType"/>
    <Datatype abbreviatedIRI="xsd:string"/>
</DataPropertyRange>
<DataPropertyRange>
    <DataProperty IRI="#hasCountry"/>
    <Datatype abbreviatedIRI="xsd:string"/>
</DataPropertyRange>
<DataPropertyRange>
    <DataProperty IRI="#hasDataType"/>
    <Datatype abbreviatedIRI="xsd:string"/>
</DataPropertyRange>
<DataPropertyRange>
    <DataProperty IRI="#hasDescription"/>
    <Datatype abbreviatedIRI="xsd:string"/>
</DataPropertyRange>
<DataPropertyRange>
    <DataProperty IRI="#hasFacebookIdentifier"/>
    <Datatype abbreviatedIRI="xsd:string"/>
</DataPropertyRange>
<DataPropertyRange>
    <DataProperty IRI="#hasGenus"/>
    <Datatype abbreviatedIRI="xsd:string"/>
</DataPropertyRange>
<DataPropertyRange>
    <DataProperty IRI="#hasMaxCardinality"/>
    <Datatype abbreviatedIRI="xsd:integer"/>
</DataPropertyRange>
<DataPropertyRange>
    <DataProperty IRI="#hasMinCardinality"/>
    <Datatype abbreviatedIRI="xsd:integer"/>
</DataPropertyRange>
<DataPropertyRange>
    <DataProperty IRI="#hasNAICSCode"/>
    <Datatype abbreviatedIRI="xsd:integer"/>
</DataPropertyRange>
<DataPropertyRange>
```

```
    <DataProperty IRI="#hasName"/>
    <Datatype abbreviatedIRI="rdfs:Literal"/>
</DataPropertyRange>
<DataPropertyRange>
    <DataProperty IRI="#hasPointOfContact"/>
    <Datatype abbreviatedIRI="xsd:string"/>
</DataPropertyRange>
<DataPropertyRange>
    <DataProperty IRI="#hasPostalCode"/>
    <Datatype abbreviatedIRI="xsd:string"/>
</DataPropertyRange>
<DataPropertyRange>
    <DataProperty IRI="#hasSemanticType"/>
    <Datatype abbreviatedIRI="xsd:string"/>
</DataPropertyRange>
<DataPropertyRange>
    <DataProperty IRI="#hasSoapStyle"/>
    <DataOneOf>
        <Literal datatypeIRI="&rdf;PlainLiteral">document</Literal>
        <Literal datatypeIRI="&rdf;PlainLiteral">rpc</Literal>
    </DataOneOf>
</DataPropertyRange>
<DataPropertyRange>
    <DataProperty IRI="#hasSoapTransport"/>
    <Datatype abbreviatedIRI="xsd:string"/>
</DataPropertyRange>
<DataPropertyRange>
    <DataProperty IRI="#hasSpecies"/>
    <Datatype abbreviatedIRI="xsd:string"/>
</DataPropertyRange>
<DataPropertyRange>
    <DataProperty IRI="#hasStateOrProvince"/>
    <Datatype abbreviatedIRI="xsd:string"/>
</DataPropertyRange>
<DataPropertyRange>
    <DataProperty IRI="#hasStatus"/>
    <Datatype abbreviatedIRI="xsd:string"/>
</DataPropertyRange>
<DataPropertyRange>
    <DataProperty IRI="#hasStreetAddress"/>
    <Datatype abbreviatedIRI="xsd:string"/>
</DataPropertyRange>
<DataPropertyRange>
    <DataProperty IRI="#hasURI"/>
    <Datatype abbreviatedIRI="xsd:anyURI"/>
</DataPropertyRange>
<DataPropertyRange>
    <DataProperty IRI="#hasURL"/>
    <Datatype abbreviatedIRI="xsd:string"/>
</DataPropertyRange>
<DataPropertyRange>
    <DataProperty IRI="#hasYelpIdentifier"/>
    <Datatype abbreviatedIRI="xsd:string"/>
</DataPropertyRange>
```

```
    <DataPropertyRange>
        <DataProperty IRI="#isDescribedByOntology"/>
        <Datatype abbreviatedIRI="xsd:anyURI"/>
    </DataPropertyRange>
    <DisjointDataProperties>
        <DataProperty IRI="#hasGenus"/>
        <DataProperty IRI="#hasSpecies"/>
    </DisjointDataProperties>
    <AnnotationAssertion>
        <AnnotationProperty abbreviatedIRI="rdfs:comment"/>
        <IRI>#Animate</IRI>
        <Literal datatypeIRI="&rdf;PlainLiteral">An Animate
PhysicalResource is a biological entity that an Operation acts upon.
these can be unambiguously described by the genus and species of the
organism. Animate Resrouces may requrie specicail consideration during
service invocation (e.g., transporting a living animal may require
environmental controls not needed when transporting blocks of
wood).</Literal>
    </AnnotationAssertion>
    <AnnotationAssertion>
        <AnnotationProperty abbreviatedIRI="rdfs:comment"/>
        <IRI>#Binding</IRI>
        <Literal datatypeIRI="&rdf;PlainLiteral">Binding is derived
from the Binding as specified in the WSDL standard. It is the means by
which a service consumer connects to the endpoint providing a given
service. It is extended here to accmodate connections to non-web
services.</Literal>
    </AnnotationAssertion>
    <AnnotationAssertion>
        <AnnotationProperty abbreviatedIRI="rdfs:comment"/>
        <IRI>#Delete</IRI>
        <Literal datatypeIRI="&rdf;PlainLiteral">Delete corresponds
directly to the DELETE method as defined by the HTTP
standard.</Literal>
    </AnnotationAssertion>
    <AnnotationAssertion>
        <AnnotationProperty abbreviatedIRI="rdfs:comment"/>
        <IRI>#Element</IRI>
        <Literal datatypeIRI="&rdf;PlainLiteral">An Element is an
atomic portion of a VirtualResource; it is an atomic data element. it
may stand alone or it may be combined with other Elements to form a
Message. It corresponds to the Element defined in the WSDL
standard.</Literal>
    </AnnotationAssertion>
    <AnnotationAssertion>
        <AnnotationProperty abbreviatedIRI="rdfs:comment"/>
        <IRI>#Get</IRI>
        <Literal datatypeIRI="&rdf;PlainLiteral">Get corresponds
directly to the GET method as defined by the HTTP standard.</Literal>
    </AnnotationAssertion>
    <AnnotationAssertion>
        <AnnotationProperty abbreviatedIRI="rdfs:comment"/>
        <IRI>#Inanimate</IRI>
```

```
        <Literal datatypeIRI="&rdf;PlainLiteral">An Inanimate
PhysicalResource is a tangible Resource that an Operation has some
effect on. It is distinct from an Animate PhysicalResource in that it
will generally not require the special considerations afforded to a
living being during service invocation.</Literal>
    </AnnotationAssertion>
    <AnnotationAssertion>
        <AnnotationProperty abbreviatedIRI="rdfs:comment"/>
        <IRI>#Message</IRI>
        <Literal datatypeIRI="&rdf;PlainLiteral">A Message is a complex
VirtualResource composed of one or more Elements. It corresponds to the
Message defined in the WSDL standard.</Literal>
    </AnnotationAssertion>
    <AnnotationAssertion>
        <AnnotationProperty abbreviatedIRI="rdfs:comment"/>
        <IRI>#NetworkBinding</IRI>
        <Literal datatypeIRI="&rdf;PlainLiteral">NetworkBinding builds
upon Binding as defined in teh WSDL specification with the intention of
expanding it beyond the SOAP-based definition to account for binding to
the endpoint for any service that can be offered across a
network.</Literal>
    </AnnotationAssertion>
    <AnnotationAssertion>
        <AnnotationProperty abbreviatedIRI="rdfs:comment"/>
        <IRI>#Notification</IRI>
        <Literal datatypeIRI="&rdf;PlainLiteral">Notification
corresponds directly to the Notification operation as defined by the
WSDL standard.</Literal>
    </AnnotationAssertion>
    <AnnotationAssertion>
        <AnnotationProperty abbreviatedIRI="rdfs:comment"/>
        <IRI>#OneWay</IRI>
        <Literal datatypeIRI="&rdf;PlainLiteral">OneWay corresponds
directly to the OneWay operation as defined by the WSDL
standard.</Literal>
    </AnnotationAssertion>
    <AnnotationAssertion>
        <AnnotationProperty abbreviatedIRI="rdfs:comment"/>
        <IRI>#Operation</IRI>
        <Literal datatypeIRI="&rdf;PlainLiteral">An Operation is the
basic functional unit of a service description. Operations are defined
by the StateChagne they effect on a Resource. The actual effect may be
nothing more than accepting some data elements as input and returning
corresponding data elements as output (e.g., returning a weather
forecast for a given location).

Operations are subdivided into the same categories as SOAP-based web
services; all interactions of any service provider and any service
consumer can be categorized in one of those ways.</Literal>
    </AnnotationAssertion>
    <AnnotationAssertion>
        <AnnotationProperty abbreviatedIRI="rdfs:comment"/>
        <IRI>#PhysicalBinding</IRI>
```

```
        <Literal datatypeIRI="&rdf;PlainLiteral">PhysicalBinding
extends the concept of a Binding as defined by WSDL to encompass
connections to the offeror of a service reqruies physical interaction
among the participants. For example, making use of the services of a
plumber to fix a leaky pipe requires the plumber to be physically
present at the location of the leak.</Literal>
    </AnnotationAssertion>
    <AnnotationAssertion>
        <AnnotationProperty abbreviatedIRI="rdfs:comment"/>
        <IRI>#PhysicalResource</IRI>
        <Literal datatypeIRI="&rdf;PlainLiteral">PhysicalResource is a
subtype of Resource that describes tangible objects an Operation may
act upon. Both web services and physical services may act upon a
PhysicalResource.</Literal>
    </AnnotationAssertion>
    <AnnotationAssertion>
        <AnnotationProperty abbreviatedIRI="rdfs:comment"/>
        <IRI>#Post</IRI>
        <Literal datatypeIRI="&rdf;PlainLiteral">Post corresponds
directly to the POST method as defined by the HTTP standard.</Literal>
    </AnnotationAssertion>
    <AnnotationAssertion>
        <AnnotationProperty abbreviatedIRI="rdfs:comment"/>
        <IRI>#Provider</IRI>
        <Literal datatypeIRI="&rdf;PlainLiteral">Provider is the
business entity that is offering a service for use, whether a network
service or a physical service.</Literal>
    </AnnotationAssertion>
    <AnnotationAssertion>
        <AnnotationProperty abbreviatedIRI="rdfs:comment"/>
        <IRI>#Put</IRI>
        <Literal datatypeIRI="&rdf;PlainLiteral">Put corresponds
directly to the PUT method as defined by the HTTP standard.</Literal>
    </AnnotationAssertion>
    <AnnotationAssertion>
        <AnnotationProperty abbreviatedIRI="rdfs:comment"/>
        <IRI>#RequestResponse</IRI>
        <Literal datatypeIRI="&rdf;PlainLiteral">RequestResponse is
based on the RequestResponse operation as defined by the WSDL standard.
It is extended with the HTTP methods GET, POST, PUT, and DELETE. This
is primarily to account for the needs of REST services; however, many
SOAP services use these methods also. Additionally, services that are
web based but are not normally considered REST services can be expected
to use one of these methods as their basis.

Beyond its use for web-based services, the RequestResponse operation is
the method in which physical services are invoked: the prospective
consumer communicates with the prospective provider to negotitate the
provision of the desired service; this is by definition a request-
response interaction.</Literal>
    </AnnotationAssertion>
    <AnnotationAssertion>
        <AnnotationProperty abbreviatedIRI="rdfs:comment"/>
        <IRI>#Resource</IRI>
```

```
        <Literal datatypeIRI="&rdf;PlainLiteral">Resource is based on
the idea of a resource as understood in the context of REST services.
It is extended here and subclassed to accomodate the description of
resources beyond that anticiapted in the REST paradigm.

Resources are anything an Operation acts upon, whenther a physical
object the Operation may affect or an element of information that is an
input or output of an Operation in a web service. </Literal>
    </AnnotationAssertion>
    <AnnotationAssertion>
        <AnnotationProperty abbreviatedIRI="rdfs:comment"/>
        <IRI>#Service</IRI>
        <Literal datatypeIRI="&rdf;PlainLiteral">A Service is an
arbitrary collection of one or more Operations. Ideally, a Service is
composed of Operations that have some logical connection, but that is
not requried.</Literal>
    </AnnotationAssertion>
    <AnnotationAssertion>
        <AnnotationProperty abbreviatedIRI="rdfs:comment"/>
        <IRI>#SolicitResponse</IRI>
        <Literal datatypeIRI="&rdf;PlainLiteral">SolicitResponse
corresponds directly to the SolicitResponse operation as defined by the
WSDL standard.</Literal>
    </AnnotationAssertion>
    <AnnotationAssertion>
        <AnnotationProperty abbreviatedIRI="rdfs:comment"/>
        <IRI>#State</IRI>
        <Literal datatypeIRI="&rdf;PlainLiteral">State is the condition
of some Resource that an Operation acts upon. An Operation may change
the State of the Resource.</Literal>
    </AnnotationAssertion>
    <AnnotationAssertion>
        <AnnotationProperty abbreviatedIRI="rdfs:comment"/>
        <IRI>#StateChange</IRI>
        <Literal datatypeIRI="&rdf;PlainLiteral">StateChange is the
fundamental action performed by an Operation. There may be a null
StateChange for an Operation that only returns information and has no
effects on the State of a Resource (such as returning a weather report
for a given location). A StateChange is defined as teh transition
between some initial State and some final State. The exact means of the
StateChange is considered an implementation detail.</Literal>
    </AnnotationAssertion>
    <AnnotationAssertion>
        <AnnotationProperty abbreviatedIRI="rdfs:comment"/>
        <IRI>#VirtualResource</IRI>
        <Literal datatypeIRI="&rdf;PlainLiteral">A VirtualResource is
an intangible Resource that an Operation may act upon. VirtualResources
generally take the form of electronic information, often in the form of
complex Messages composed of individual Elements.</Literal>
    </AnnotationAssertion>
</Ontology>

<!-- Generated by the OWL API (version 3.4.2)
http://owlapi.sourceforge.net -->
```

APPENDIX B: CPLEX OPL OUTPUT

The ECNE optimization formalism is implemented using the IBM CPLEX Integrated Development Environment (IDE) implementation of OPL. The results of that implementation are shown in the figures below.

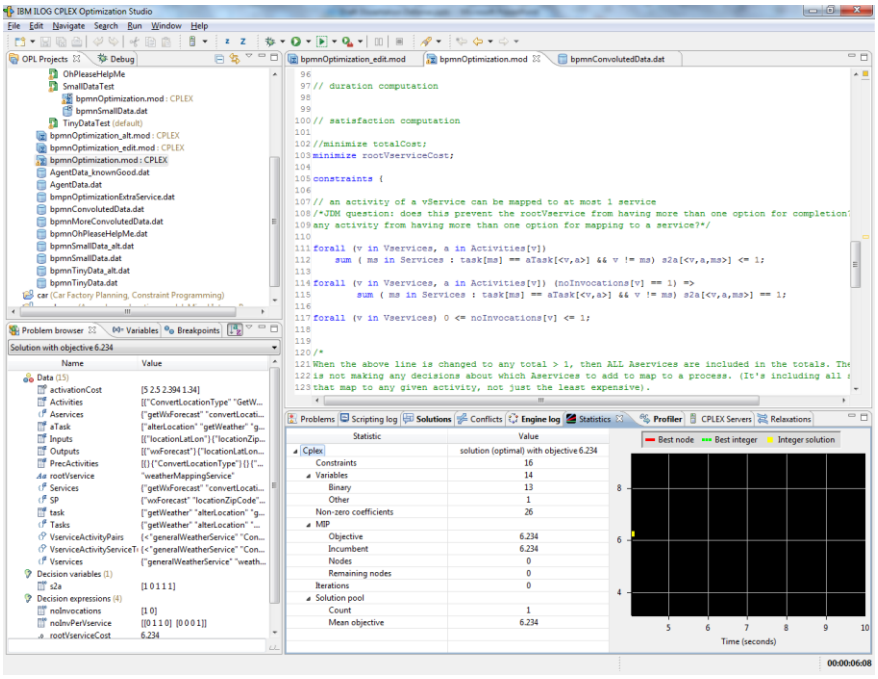

Figure 21: ECNE Implementation Results

The code listing of the ECNE model is shown in the upper right quadrant of Figure 21. Results of the optimization ncomputation are showin in the bottom right, and

details of the decision variables and the results of the decision expressions are shown in the lower-left corner. An enlarged view of this data is shown in Figure 22.

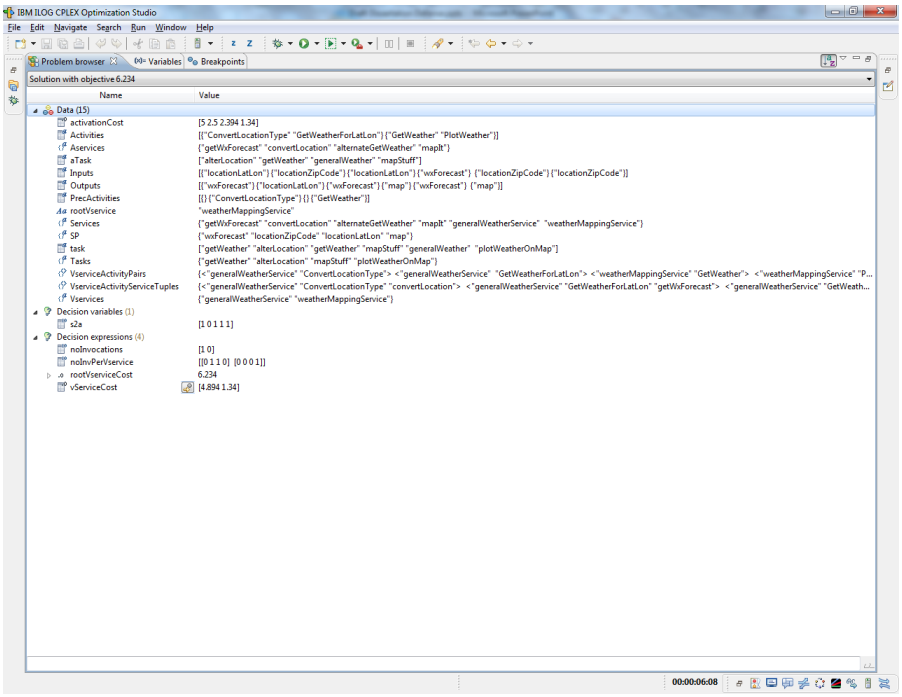

Figure 22: Details of decision variables and expressions

REFERENCES

1. Erl T. Service-Oriented Architecture (SOA) Concepts, Technology and Design. Prentice Hall; 2005.

2. Christensen E, Curbera F, Meredith G, Weerawarana S, editors. Web Services Description Language (WSDL) 1.1 [Internet]. W3C; 2001. Available from: http://www.w3.org/TR/wsdl

3. Chinnici R, Moreau J-J, Ryman A, Weerawarana S, editors. Web Services Description Language (WSDL) Version 2.0 [Internet]. W3C; 2007. Available from: http://www.w3.org/TR/wsdl20/

4. Akkiraju R, Farrell J, Miller JA, Nagarajan M, Sheth A, Verma K. Web Service Semantics - WSDL-S [Internet]. Yorktown Heights, NY: Thomas J. Watson Research Center; 2006 Jan. Report No.: RC23854 (W0601-132). Available from: http://domino.research.ibm.com/library/cyberdig.nsf/papers/EF9FE52551FB21DC8525710D005A8480/$File/rc23854.pdf

5. Kopecky J, Vitvar T, Bournez C, Farrell J. SAWSDL: Semantic Annotations for WSDL and XML Schema. IEEE Internet Comput. 2007 Dec;11(6):60 –67.

6. Martin D, Paolucci M, McIlraith S, Burstein M, McDermott D, McGuinness D, et al. Bringing Semantics to Web Services: The OWL-S Approach. Semantic Web Services and Web Process Composition [Internet]. 2005 [cited 2009 Aug 9]. p. 26–42. Available from: http://www.springerlink.com/content/rl5r1c8v64xvf0r8

7. Fielding R. Architectural Styles and the Design of Network-based Software Architectures [Internet] [Doctoral Dissertation]. [Irvine, CA]: University of California, Irvine; 2000. Available from: http://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm

8. Hadley MJ. Web application description language (WADL). Mountain View, CA, USA: Sun Microsystems, Inc.; 2006.

9. Business Process Model and Notation, Version 1.2 [Internet]. Object Management Group; 2009. Available from: http://www.omg.org/spec/BPMN/1.2/

10. Business Process Model and Notation (BPMN) Version 2.0 [Internet]. Object Management Group; 2011. Available from: http://www.omg.org/spec/BPMN/2.0/

11. Feier C, Polleres A, Dumitru R, Domingue J, Stollberg M, Fensel D. Towards intelligent web services: the web service modeling ontology (WSMO) [Internet]. 2005 [cited 2012 Sep 16]. Available from: http://oro.open.ac.uk/23147/

12. Fensel D, Bussler C. The Web Service Modeling Framework WSMF. Electron Commer Res Appl. 2002;1(2):113–37.

13. Patil AA, Oundhakar SA, Sheth AP, Verma K. Meteor-s web service annotation framework. Proceedings of the 13th international conference on World Wide Web [Internet]. New York, NY, USA: ACM; 2004 [cited 2009 Aug 11]. p. 553–62. Available from: http://portal.acm.org/citation.cfm?id=988672.988747&coll=portal&dl=ACM&type=series&idx=SERIES968&part=series&WantType=Proceedings&title=WWW

14. Verma K, Gomadam K, Sheth AP, Miller JA, Wu Z. The METEOR-S approach for configuring and executing dynamic web processes. 2005.

15. OMG MOF 2 XMI Mapping Specification v2.4.1 [Internet]. Object Mana; 2013. Available from: http://www.omg.org/spec/XMI/2.4.1/

16. Klusch M, Kapahnke P. Semantic Web Service Selection with SAWSDL-MX. Service Matchmaking and Resource Retrieval in the Semantic Web (SMR2 2008). Karlsruhe, Germany; 2008. p. 3–17.

17. Klusch M, Kapahnke P, Zinnikus I. SAWSDL-MX2: A Machine-Learning Approach for Integrating Semantic Web Service Matchmaking Variants. IEEE International Conference on Web Services, 2009 ICWS 2009. 2009. p. 335 –342.

18. Klusch M, Fries B, Sycara K. Automated semantic web service discovery with OWLS-MX. Proceedings of the fifth international joint conference on Autonomous agents and multiagent systems [Internet]. Hakodate, Japan: ACM; 2006 [cited 2009 Aug 11]. p. 915–22. Available from: http://portal.acm.org/citation.cfm?id=1160796

19. Klusch M, Fries B, Sycara K. OWLS-MX: A hybrid Semantic Web service matchmaker for OWL-S services. Web Semant Sci Serv Agents World Wide Web. 2009 Apr;7(2):121–33.

20. Klusch M, Kapahnke P. Adaptive signature-based semantic selection of services with OWLS-MX3. Multiagent Grid Syst. 2012 Jan 1;8(1):69–82.

21. Klusch M, Kapahnke P. OWLS-MX3: An Adaptive Hybrid Semantic Service Matchmaker for OWL-S. Proceedings of the 3rd International SMR2 2009

Workshop on Service Matchmaking and Resource Retrieval in the Semantic Web [Internet]. Washington, DC, USA: CEUR Workshop Proceedings; 2009. Available from: http://ceur-ws.org/Vol-525/paper3.pdf

22. Alasti M, Neekzad B, Hui J, Vannithamby R. Quality of service in WiMAX and LTE networks [Topics in Wireless Communications]. Commun Mag IEEE. 2010;48(5):104–11.

23. Luo H, Shyu M-L. Quality of service provision in mobile multimedia-a survey. Hum-Centric Comput Inf Sci. 2011;1(1):1–15.

24. Menascé DA, Ewing JM, Gomaa H, Malex S, Sousa JP. A framework for utility-based service oriented design in SASSY. Proceedings of the first joint WOSP/SIPEW international conference on Performance engineering [Internet]. San Jose, California, USA: ACM; 2010 [cited 2010 May 25]. p. 27–36. Available from: http://portal.acm.org/citation.cfm?id=1712605.1712612

25. Mabrouk NB, Beauche S, Kuznetsova E, Georgantas N, Issarny V. QoS-Aware Service Composition in Dynamic Service Oriented Environments. In: Bacon JM, Cooper BF, editors. Middleware 2009 [Internet]. Springer Berlin Heidelberg; 2009 [cited 2014 Jan 2]. p. 123–42. Available from: http://link.springer.com/chapter/10.1007/978-3-642-10445-9_7

26. Mabrouk NB, Georgantas N, Issarny V. A Semantic End-to-end QoS Model for Dynamic Service Oriented Environments. Proceedings of the 2009 ICSE Workshop on Principles of Engineering Service Oriented Systems [Internet]. Washington, DC, USA: IEEE Computer Society; 2009 [cited 2014 Jan 2]. p. 34–41. Available from: http://dx.doi.org/10.1109/PESOS.2009.5068817

27. Mabrouk NB, Georgantas N, Issarny V. QoS-aware Service-oriented Middleware for Pervasive Environments. Proceedings of the 10th ACM/IFIP/USENIX International Conference on Middleware [Internet]. New York, NY, USA: Springer-Verlag New York, Inc.; 2009 [cited 2014 Jan 2]. p. 27:1–27:2. Available from: http://dl.acm.org/citation.cfm?id=1656980.1657017

28. Yu T, Lin K-J. Service Selection Algorithms for Composing Complex Services with Multiple QoS Constraints. In: Benatallah B, Casati F, Traverso P, editors. Service-Oriented Computing - ICSOC 2005 [Internet]. Springer Berlin Heidelberg; 2005 [cited 2013 Sep 21]. p. 130–43. Available from: http://link.springer.com/chapter/10.1007/11596141_11

29. McIlraith SA, Son TC, Honglei Zeng. Semantic Web Services. Intell Syst IEEE. 2001;16(2):46–53.

30. Paolucci M, Srinivasan N, Sycara K, Nishimura T. Towards a Semantic Choreography of Web Services: From WSDL to DAML-S. Proceedings of the International Conference on Web Services (ICWS 2003). 2003. p. 22–6.

31. Sycara K, Paolucci M, Ankolekar A, Srinivasan N. Automated Discovery, Interaction and Composition of Semantic Web Services. Web Semant Sci Serv Agents World Wide Web [Internet]. 2011 Mar 8;1(1). Available from: http://imap.websemanticsjournal.org/index.php/ps/article/view/25

32. Larvet P, Christophe B, Pastor A. Semantization of Legacy Web Services: From WSDL to SAWSDL. Internet and Web Applications and Services, 2008 ICIW '08 Third International Conference on. 2008. p. 130–5.

33. Recker JC, Mendling J. On the Translation between BPMN and BPEL: Conceptual Mismatch between Process Modeling Languages. 18th Int Conf Adv Inf Syst Eng Proc Workshop Dr Consort. 2006;521–32.

34. Juric MB. Business Process Execution Language for Web Services BPEL and BPEL4WS 2nd Edition [Internet]. Packt Publishing; 2006 [cited 2010 Mar 26]. Available from: http://portal.acm.org/citation.cfm?id=1199048&CFID=83594666&CFTOKEN=79603150

35. Kloppmann M, Koenig D, Laymann F, Pfau G, Rickayzen A, von Riegen C, et al. WS-BPEL Extension for People – BPEL4People: A Joint White Paper by IBM and SAP [Internet]. 2007. Available from: http://democritique.org/IT/Documents/BPEL/BPEL4People_white_paper.pdf

36. Clement L, Koenig D, Mehta V, Mueller R, Rangaswamy R, Rowley M, et al., editors. WS-BPEL Extension for People (BPEL4People) Specification [Internet]. OASIS; 2010. Available from: http://docs.oasis-open.org/bpel4people/bpel4people-1.1.html

37. Clement L, Koenig D, Mehta V, Mueller R, Rangaswamy R, Rowley M, et al., editors. Web Services – Human Task (WS-HumanTask) Specification [Internet]. 2010. Available from: http://docs.oasis-open.org/bpel4people/ws-humantask-1.1.html

38. Cardoso J, Sheth A. Semantic E-Workflow Composition. J Intell Inf Syst. 2003 Nov;21:191–225.

39. McDowall J, Kerschberg L. Agent Negotiation Strategies for Composing Service Workflows. Washington, DC, USA; 2012.

40. Alodhaibi K. Decision-Guided Recommenders With Composite Alternatives [Internet]. [Fairfax, VA]: George Mason University; 2011. Available from: http://hdl.handle.net/1920/6591

41. Brodsky A, Morgan Henshaw S, Whittle J. CARD: a decision-guidance framework and application for recommending composite alternatives. Proceedings of the 2008 ACM conference on Recommender systems [Internet]. New York, NY, USA: ACM; 2008 [cited 2012 Sep 23]. p. 171–8. Available from: http://doi.acm.org/10.1145/1454008.1454037

42. Brodsky A, Wang XS. Decision-Guidance Management Systems (DGMS): Seamless Integration of Data Acquisition, Learning, Prediction and Optimization. Hawaii International Conference on System Sciences, Proceedings of the 41st Annual. 2008. p. 71.

43. Alodhaibi K, Brodsky A, Mihaila GA. COD: Iterative Utility Elicitation for Diversified Composite Recommendations. 2010 43rd Hawaii International Conference on System Sciences (HICSS). 2010. p. 1 –10.

44. McDowall J, Kerschberg L. Optimizing Service Selection in Dynamic Workflow Composition: Using Social Media to Develop Recommendations. San Jose, California, USA; 2012.

45. McDowall J, Kerschberg L. Leveraging Social Networks to Improve Service Selection in Workflow Composition. Istanbul, Turkey; 2012.

46. Floyd RW. Algorithm 97: shortest path. Commun ACM. 1962;5(6):345.

47. McDowall J, Brodsky A, Kerschberg L. A Formal Model for Optimizing Dynamic Service Composition. Richmond, VA; 2015.

48. Mabrouk NB, Beauche S, Kuznetsova E, Georgantas N, Issarny V. QoS-Aware Service Composition in Dynamic Service Oriented Environments. In: Bacon JM, Cooper BF, editors. Middleware 2009 [Internet]. Springer Berlin Heidelberg; 2009 [cited 2013 Mar 9]. p. 123–42. Available from: http://link.springer.com/chapter/10.1007/978-3-642-10445-9_7

49. Mabrouk NB, Georgantas N, Issarny V. QoS-aware service-oriented middleware for pervasive environments. Proceedings of the 10th ACM/IFIP/USENIX International Conference on Middleware [Internet]. New York, NY, USA: Springer-Verlag New York, Inc.; 2009 [cited 2013 Mar 9]. p. 27:1–27:2. Available from: http://dl.acm.org/citation.cfm?id=1656980.1657017

50. FIPA Contract Net Interaction Protocol Specification [Internet]. Foundation for Intelligent Physical Agents; 2002. Available from: http://www.fipa.org/specs/fipa00029/SC00029H.pdf

51. Muthaiyah S, Barbulescu M, Kerschberg L. An Improved Matching Algorithm for Developing a Consistent Knowledge Model across Enterprises Using SRS and SWRL. Hawaii International Conference on System Sciences. Los Alamitos, CA, USA: IEEE Computer Society; 2009. p. 1–9.

52. Rumbaugh J, Jacobson I, Booch G. Unified Modeling Language Reference Manual, The (2nd Edition). Pearson Higher Education; 2004.

53. Zur Mehlan M. Enterprise Architecture based on Design Primitives and Patterns: Guidelines for the Design and Development of Event-Trace Descriptions (DoDAF OV-6c) using BPMN [Internet]. Business Transformation Agency; 2009. Available from: http://dodcio.defense.gov/Portals/0/Documents/DODAF/Primitives_OV-6c_Guidelines.pdf

54. Zur Mehlan M, Recker J. How Much Language Is Enough? Theoretical and Practical Use of the Business Process Modeling Notation. Advanced Information Systems Engineering. 2008.

55. McGuinness D, van Harmelen F, editors. OWL Web Ontology Language Overview [Internet]. W3C; 2004. Available from: http://www.w3.org/TR/owl-features/

56. Baader F, McGuinness D, Nardi D, Patel-Schneider PF, editors. The Description Logic Handbook: Theory, Implementation, and Applications. Cambridge University Press; 2003. 568 p.

57. Lassila O, Swick RR, Wide W, Consortium W. Resource Description Framework (RDF) Model and Syntax Specification. 1998.

58. Horrocks I, Patel-Schneider PF, Boley H, Tabet S, Grosof B, Dean M. SWRL: A Semantic Web Rule Language Combining OWL and RuleML. W3C Member Submission; 2004.

59. Muthaiyah S, Kerschberg L. A Hybrid Ontology Mediation Approach for the Semantic Web. Int J E-Bus Res. 2008;4(4):79–91.

60. Muthaiyah S, Barbulescu M, Kerschberg L. A Hybrid Similarity Matching Algorithm for Mapping and Upgrading Ontologies via a Multi-Agent System. Heraklion, Crete Island, Greece; 2008.

61. Linné C von, Gmelin JF. Systema Naturae: Per Regna Tria Naturae, Secundum Classes, Ordines, Genera, Species, Cum Characteribus, Differentiis, Synonomis, Locis. 13th ed. Leipzig: G.E. Beer; 1788.

62. Noy NF, Crubézy M, Fergerson RW, Knublauch H, Tu SW, Vendetti J, et al. Protege-2000: an open-source ontology-development and knowledge-acquisition environment. AMIA Annu Symp Proc. 2003;953:953.

63. Horridge M, Drummond N, Goodwin J, Rector AL, Stevens R, Wang H. The Manchester OWL Syntax. OWLed [Internet]. 2006 [cited 2014 Mar 22];216. Available from: http://owl1-1.googlecode.com/svn-history/r670/trunk/www.webont.org/owled/2006/acceptedLong/submission_9.pdf

64. Suguri H. A standardization effort for agent technologies: The Foundation for Intelligent Physical Agents and its activities. Proceedings of the 32nd Annual Hawaii International Conference on Systems Sciences, 1999 HICSS-32. 1999. p. 10 pp.–.

BIOGRAPHY

John D. McDowall graduated from Monticello High School, Monticello, New York, in 1985. He received his Bachelor of Science from the United States Naval Academy in 1989. He served as a CH-53E pilot in the United States Marine Corps and received his Master of Science in Computer Information Systems from Boston University in 1999.