ATHABASCA UNIVERSITY

SEMANTIC WEB SERVICES-BASED REASONING IN THE DESIGN OF SOFTWARE PRODUCT LINES

BY

JAMES JEFFREY RUSK

An integrated project submitted in partial fulfillment

Of the requirements for the degree of

MASTER OF SCIENCE in INFORMATION SYSTEMS

Athabasca, Alberta
August, 2009

© James Jeffrey Rusk, 2009
ATHABASCA UNIVERSITY

The undersigned certify that they have read and recommend for acceptance the integrated project

"SEMANTIC WEB-BASED REASONING IN THE DESIGN OF SOFTWARE PRODUCT LINES"

submitted by JAMES JEFFREY RUSK in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE in INFORMATION SYSTEMS.

___________________________
Dragan Gašević, Ph.D.
Supervisor

___________________________
Jon Dron, Ph.D.
Chair

___________________________
Marek Hatala, Ph.D.
Examiner

Date: _______________________

ii
ABSTRACT

Software Product Line (SPL) practices have been found to reduce both the cost of software development and the number of defects. However, there are a number of impediments to successful implementation of SPL, especially when considering service oriented architectures (SOA). Enabling product architectures to configure themselves autonomously at runtime is an emerging area of SPL research. One of the most promising technologies available to enable such runtime dynamism is the Semantic Web. Using a Semantic Web service description language to encode product configurations, the Semantic Web may better enable service discovery, composition, and invocation while retaining the benefits of SPL. Adopting the Feature Model to represent Web services in a SPL and using the Web Service Modeling Ontology (WSMO) to encode SPL configurations, this project leverages ontology-based reasoning to provide information to the developer regarding how the proposed composition would function with the real-world services selected. The development of a set of mappings between the feature model structure and WSMO elements supports a transformation from product configuration to Semantic Web service descriptions in WSMO. The WSMX execution environment deploys and tests the composition, providing ontology-based reasoning to the developer. Measures of success in relation to this project include testing the accuracy of the mapping possible between the two formalisms, verifying the degree of automation which can be supported during transformation, the subsequent precision of feature discovery, as well as exploring the support or guidance that the ontology can provide to feature modeling. Overall, as a feature model can be considered a view on an ontology, and a SOA can be represented with WSMO elements, this work demonstrates what can be done within this feature model view and what support or guidance ontology-based reasoning can provide while working within this view.
ACKNOWLEDGEMENTS

I sincerely appreciate the support of my integrated project supervisor, Dragan Gašević, who introduced me to these concepts during COMP 610 Software Engineering and provided a great deal of guidance when I chose to explore these ideas further through the integrated project. His assistance, not just in terms of research direction but also regarding funding and dissemination opportunities, ensured the overall experience of this integrated project was very worthwhile and memorable. I would also like to acknowledge the generous support of Athabasca University, specifically the Student Awards Unit and University Research Services. Other researchers who provided useful insight to me during this work were the Software Systems Reading Group participants with the Laboratory for Ontological Research (LORE) based out of Simon Fraser University. I would like to thank both my previous employers, the Nunavut Planning Commission and the Nunavut Impact Review Board, for encouraging me to pursue further education while working full-time. Finally, this work would not have been possible without the support and patience of my family. The patience of my young children, Alyssa, Nathan, and Justin, were especially taxed and the greatest acknowledgement I can offer is to them.
# TABLE OF CONTENTS

<table>
<thead>
<tr>
<th>CHAPTER 1</th>
<th>INTRODUCTION</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.1</td>
<td>Background</td>
<td>1</td>
</tr>
<tr>
<td>1.2</td>
<td>The Research Question</td>
<td>3</td>
</tr>
<tr>
<td>1.3</td>
<td>Research Contributions</td>
<td>3</td>
</tr>
<tr>
<td>1.4</td>
<td>Research Overview</td>
<td>4</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>CHAPTER 2</th>
<th>REVIEW OF RELATED LITERATURE</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1</td>
<td>Service Oriented Architecture</td>
<td>5</td>
</tr>
<tr>
<td>2.2</td>
<td>The Semantic Web</td>
<td>12</td>
</tr>
<tr>
<td>2.3</td>
<td>Software Product Lines</td>
<td>16</td>
</tr>
<tr>
<td>2.4</td>
<td>Integration of SOA and SPL</td>
<td>20</td>
</tr>
<tr>
<td>2.5</td>
<td>Model Driven Development</td>
<td>25</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>CHAPTER 3</th>
<th>METHODOLOGY</th>
<th>27</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.1</td>
<td>Feature Modeling of Service Oriented Architectures</td>
<td>28</td>
</tr>
<tr>
<td>3.2</td>
<td>Composition in the Web Service Modeling Ontology</td>
<td>29</td>
</tr>
<tr>
<td>3.3</td>
<td>Transformation Between SOA and SPL Views</td>
<td>33</td>
</tr>
<tr>
<td>3.4</td>
<td>Verification, Reasoning, and Analysis</td>
<td>38</td>
</tr>
<tr>
<td>3.5</td>
<td>Prototyping and Testing Framework</td>
<td>41</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>CHAPTER 4</th>
<th>DETAILED DESIGN</th>
<th>51</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.1</td>
<td>Representation of Variability</td>
<td>52</td>
</tr>
<tr>
<td>4.2</td>
<td>Mapping and Transformation</td>
<td>55</td>
</tr>
<tr>
<td>4.3</td>
<td>Manual Intervention</td>
<td>64</td>
</tr>
<tr>
<td>Section</td>
<td>Title</td>
<td>Page</td>
</tr>
<tr>
<td>---------</td>
<td>------------------------------------------------------------</td>
<td>------</td>
</tr>
<tr>
<td>4.4</td>
<td>Configuration of WSMX</td>
<td>68</td>
</tr>
<tr>
<td>4.5</td>
<td>Reasoning</td>
<td>69</td>
</tr>
<tr>
<td>CHAPTER V</td>
<td>DEMONSTRATION</td>
<td>75</td>
</tr>
<tr>
<td>5.1</td>
<td>Running Case</td>
<td>75</td>
</tr>
<tr>
<td>5.2</td>
<td>Web Services</td>
<td>76</td>
</tr>
<tr>
<td>5.3</td>
<td>Feature Models and Product Configurations</td>
<td>80</td>
</tr>
<tr>
<td>5.4</td>
<td>Transformation</td>
<td>84</td>
</tr>
<tr>
<td>5.5</td>
<td>Manual Intervention</td>
<td>87</td>
</tr>
<tr>
<td>5.6</td>
<td>Reasoning Output</td>
<td>92</td>
</tr>
<tr>
<td>CHAPTER 6</td>
<td>RESULTS AND ANALYSIS</td>
<td>100</td>
</tr>
<tr>
<td>6.1</td>
<td>Results</td>
<td>100</td>
</tr>
<tr>
<td>6.2</td>
<td>Evaluation</td>
<td>104</td>
</tr>
<tr>
<td>6.3</td>
<td>Comparison</td>
<td>111</td>
</tr>
<tr>
<td>CHAPTER 7</td>
<td>CONCLUSIONS AND RECOMMENDATIONS</td>
<td>118</td>
</tr>
<tr>
<td>7.1</td>
<td>Conclusions</td>
<td>118</td>
</tr>
<tr>
<td>7.2</td>
<td>Recommendations</td>
<td>120</td>
</tr>
<tr>
<td>7.3</td>
<td>Suggestions for Further Research</td>
<td>121</td>
</tr>
<tr>
<td>REFERENCES</td>
<td></td>
<td>124</td>
</tr>
<tr>
<td>APPENDIX A</td>
<td></td>
<td>137</td>
</tr>
<tr>
<td>APPENDIX B</td>
<td></td>
<td>138</td>
</tr>
<tr>
<td>APPENDIX C</td>
<td></td>
<td>139</td>
</tr>
<tr>
<td>APPENDIX D</td>
<td></td>
<td>140</td>
</tr>
</tbody>
</table>
LIST OF TABLES

Table 1. Summary of Basic Mapping Rules \( \ldots \) \( \ldots \) 59

Table 2. Summary of Origin of WSML Web Service Elements \( \ldots \) \( \ldots \) 67

Table 3. Average Percentage of Automatically Generated WSML \( \ldots \) \( \ldots \) 105
LIST OF FIGURES

Figure 1. Research Domains and Targets ........................................ 2
Figure 2. The Find-Bind-Execute Paradigm ..................................... 6
Figure 3. Orchestration and Choreography .................................... 8
Figure 4. Top Level Elements of WSMO ........................................ 15
Figure 5. General Methodology Outline ........................................ 27
Figure 6. Mappings and Transformations ...................................... 35
Figure 7. Feature Model Metamodel ............................................. 36
Figure 8. Abbreviated WSMO Metamodel ..................................... 36
Figure 9. Feature Model in the Feature Model Plugin ....................... 44
Figure 10. Process Supported by the Detailed Design ........................ 51
Figure 11. Basic Feature Model Methodology ................................. 52
Figure 12. Example Feature Model .............................................. 54
Figure 13. Example Product Configuration .................................... 55
Figure 14. Model, Metamodel, and Metametamodel in .fmp File .......... 61
Figure 15. Steps of Demonstration ............................................. 75
Figure 16. Demonstration Feature Model ...................................... 81
Figure 17. Demonstration Product Configuration ............................. 83
Figure 18. Composition Logic .................................................... 83
Figure 19. Output Ecore WSMO Model ......................................... 84
<table>
<thead>
<tr>
<th>Listing</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Feature Model XML Export</td>
<td>45</td>
</tr>
<tr>
<td>2</td>
<td>Feature Group Helper Functions</td>
<td>62</td>
</tr>
<tr>
<td>3</td>
<td>Metamodel Helper Function</td>
<td>62</td>
</tr>
<tr>
<td>4</td>
<td>Calling Helper Functions</td>
<td>63</td>
</tr>
<tr>
<td>5</td>
<td>Lazy Rule</td>
<td>63</td>
</tr>
<tr>
<td>6</td>
<td>Direction in <code>config.properties</code> File to Available WSMO Resources</td>
<td>68</td>
</tr>
<tr>
<td>7</td>
<td>Example <code>AchieveGoalExecutionSemanticsTest.java</code></td>
<td>70</td>
</tr>
<tr>
<td>8</td>
<td>Examples as Loaded into WSMX</td>
<td>70</td>
</tr>
<tr>
<td>9</td>
<td>Failure to Parse</td>
<td>71</td>
</tr>
<tr>
<td>10</td>
<td>WSMX Output with Lack of Discovery</td>
<td>71</td>
</tr>
<tr>
<td>11</td>
<td>Successful Discovery</td>
<td>72</td>
</tr>
<tr>
<td>12</td>
<td>Data Mediation</td>
<td>73</td>
</tr>
<tr>
<td>13</td>
<td>Abbreviated Choreography Output</td>
<td>73</td>
</tr>
<tr>
<td>14</td>
<td>Successful Output Instance Data</td>
<td>73</td>
</tr>
<tr>
<td>15</td>
<td>Contents of Web Service Java Package Folder</td>
<td>77</td>
</tr>
<tr>
<td>16</td>
<td>WSDL File for AtlanticTech Producer Web Service</td>
<td>77</td>
</tr>
<tr>
<td>17</td>
<td>Input Message to AtlanticTech Producer Web Service</td>
<td>79</td>
</tr>
<tr>
<td>18</td>
<td>Output Message from AtlanticTech Producer Web Service</td>
<td>79</td>
</tr>
<tr>
<td>19</td>
<td>Semantic Web Service Description for AtlanticTech Web Service</td>
<td>80</td>
</tr>
<tr>
<td>20</td>
<td>Output WSML Listing of Composite Service</td>
<td>85</td>
</tr>
<tr>
<td>21</td>
<td>Manual Intervention for Payment Mediator Process</td>
<td>88</td>
</tr>
</tbody>
</table>
Listing 22. Additional States of the ASM 88
Listing 23. Web Service Description Following Manual Intervention 89
Listing 24. Goal to Discover and Invoke Demonstration Web Service 91
Listing 25. Web Services are Available to WSMX 93
Listing 26. Successful Discovery and Loading of Choreography 94
Listing 27. Groundings 94
Listing 28. Process Mediation 95
Listing 29. Invoking of Molecular Web Services 95
Listing 30. Response Ontology 97
Listing 31. Connection to Actual Web Service Refused 98
Listing 32. Inconsistent Ontologies Underlying the Composition 99
CHAPTER 1
INTRODUCTION

1.1 BACKGROUND

An Integrated Project necessarily must contain a significant element of integration in order to make a contribution which can be considered novel. The ability to integrate diverse concepts and technologies in order to achieve synergies and efficiencies is the hallmark of such research directions. The World Wide Web and its related technologies have introduced a wide variety of means by which enhanced collaboration and communication can be achieved. However, flexible and effective software engineering methods must be developed to take advantage of these opportunities. This particular research attempts to better integrate the diverse concepts and technologies present in the fields of service oriented architectures (SOA), software product lines (SPL), and model driven development (MDD) in order to achieve such synergies. While these areas of practice are diverse within the information systems domain, there do exist methodologies and technologies which can facilitate their integration. This project explores a set of methodologies and technologies which represent one of the many possible approaches to achieving such synergy between the SOA and SPL domains.

The practices associated with SPL are expected to significantly change how software development is performed and perceived (Schmid and Verlage 2002). However, there are a number of limitations within the current state of the art in SPL, especially when considering SOA. Research facilitating the ability of modern architectures, including SOA, to configure themselves autonomously at runtime is an emerging area of SPL research (Sugumaran, Park, and Kang 2006). With a great deal of research still required to achieve complete automatic interoperation between Web services in the "open" domain of SOA generally (Sycara, Paolucci, Ankolekar, and Srinivasan 2003) and with SPL
remaining a fairly closed domain, there is significant opportunity to contribute further to the integration possibilities between these two domains. With SOA being an open architecture and SPL typically a relatively closed one, there need to be some ways to bridge the gap between these two approaches before true synergies can be achieved. Figure 1 graphically illustrates the research domains and the target contributions that this work operates within. When viewing Figure 1 in the context of the above discussion, it should be clear that the areas of research within C and D are the primary targets of this work. However, in order to operate within these narrow integrative disciplines it is necessary to harness the concepts and technologies associated with both MDD and the Semantic Web. As can be seen, ontology technologies must also be leveraged to support the connection between the domains of SOA and SPL.

Figure 1. Research Domains and Targets

![Ontology Diagram]

The concept of ontology, specifically ontologies as conceived to support the Semantic Web (Berners-Lee, Hendler, and Lassila 2001), is the ideal technology to bridge the gap between these two domains of SOA and SPL so that their modeling approaches, once made interoperable, can achieve a useful level of integration. Although ontology-facilitated technologies will be shown to enable the
integration of these two disciplines, it does introduce a level of complexity which can be difficult to manage. The ability to harness the concepts and technologies associated with MDD will be shown to alleviate these difficulties and ensure that the synergies are achievable in an efficient manner.

There is a diverse array of perspectives through which to view potential integration within these realms. In this particular research it is proposed that, by applying a SPL perspective to a SOA model and utilizing the reasoning functionality that is enabled through Semantic Web technologies, the design and configuration of SOAs can be modeled through a SPL. Considering that a feature model, a popular SPL model representation, can be considered a view on an ontology (Czarnecki, Kim, and Kallberg 2006; Kim 2006), and that a SOA can be represented using ontology constructs, a key question is what can be done within this feature model view and what support or guidance can ontology-based reasoning provide while working within that view. The answers to these questions have the potential to improve both SPL and SOA singularly but, more importantly, allow the two domains to better function with each other.

1.2 THE RESEARCH QUESTION

The specific goal of this research is to evaluate the suitability of the Web Service Modeling Ontology (WSMO) in the encoding of product configurations from a software product line in such a manner as to enable reasoning approaches which facilitate higher automation of service discovery, composition, and invocation.

1.3 RESEARCH CONTRIBUTIONS

The overall contribution of this research is to the ever improving set of software engineering methods and tools which integrate the practices of SPL and SOA. Specifically, the contributions include mappings to support transformation between the SPL and SOA models of a system in addition to the verification mechanisms within the feature model view based on SWS reasoning. Furthermore,
this research also serves to evaluate further the suitability of the Web Service Modeling Ontology (WSMO) in the context of supporting SPL practices. Together, these contributions explore the accuracy of the mapping which is possible between the two formalisms, the level of automation which can be supported during transformation, the resulting precision of feature discovery, and the support or guidance that ontology-based reasoning engines can provide to feature modeling. Improvements in both SOA and SPL practices can be derived from this work.

1.4 RESEARCH OVERVIEW

This Integrated Project has been organized in seven chapters. Chapter 1 briefly introduces the background of the diverse concepts and technologies that this research seeks to further integrate. It outlines the goal of the project, the potential contributions it makes, as well as the layout of subsequent chapters. Chapter 2 is a review of the relevant literature which provides a supporting body of knowledge which this work can build upon. The diverse elements of this research require that this chapter be fairly extensive, however, the wide-ranging works all clearly contribute to the integrative nature of this research. The proposed solution and the methodologies used are discussed in Chapter 3. Chapter 4 illustrates the detailed design of the solution and its implementation. Chapter 5 demonstrates the use of the implemented solution. Results and analysis are contained within Chapter 6, in addition to a comparison with other solutions. Chapter 7 concludes the work and makes recommendations for future research.
CHAPTER 2
REVIEW OF RELATED LITERATURE

2.1 SERVICE ORIENTED ARCHITECTURE

Internet technologies, specifically the World Wide Web, have created a tremendous variety of opportunities for collaboration and communication. Virtually all sectors of business, governance, and even the personal lives of individuals have triggered an increasing demand for novel ways to take advantage of these opportunities. Service-oriented Computing is considered an emerging cross-disciplinary paradigm for distributed computing which is changing the way that software products are designed, created, and deployed. For Service-oriented Computing to be successful though, it necessarily requires implementation of a Service-Oriented Architecture (SOA). An architecture centered around services allows for collaboration and communication on a scale that was not previously possible with earlier technologies. In a SOA, the architecture and requirements of the business itself drive the design, creation, and deployment of services and can be made to serve entire "business communities" which are, in effect, groups of diverse actors that must cooperate and function within the same business area (Baglietto, Maresca, Parodi, and Zingirian 2005). The implementation of SOA can overcome numerous technical and organizational challenges including differences in network topologies and operating systems as well as common issues such as legacy content conversion (Maurizio, Sagar, Jones, Corbitt, and Girolami 2008).

For a SOA to be successful, it must make use of standard languages and protocols, in additional to utilizing open standards in a technology-neutral manner wherever possible. A SOA is typically composed of services, registries, service users, as well as proxies. It is structured to minimize coupling but maximize reuse, while implementing a find-bind-execute paradigm as shown in Figure 2.
Services in a SOA typically require the following characteristics in order to be made useful (McGovern, Tyagi, Stevens, Mathew 2003):

- Discoverable and dynamically bound
- Self-contained and modular
- Interoperable
- Loosely coupled
- Network-addressable interface
- Coarse-grained interfaces
- Location-transparent
- Composable

However, the technological and organizational realization of this architecture is still in its infancy. SOA raises unique challenges. There are a number of issues that the basic SOA does not currently address including management, service orchestration, service transaction management and coordination, and security (Papazoglou 2003). Furthermore, the increasing utilization of SOA will likely lead to further issues of service and architecture management not fully considered in earlier implementations. For example, it is certain that existing services in a SOA will be considered for assembly into future applications that the original designer of the service did not envision. As discovery and composition of increasingly complex combinations of services necessarily become more automated, there is a requirement for more mature and technically sound mechanisms for the design and development of
WEB SERVICES. The most mature realization, and by far the most common implementation, of a SOA is currently based on Web Services. While Web Services are not the only implementation of a SOA, and in fact they do not currently implement all concepts of an ideal SOA, they are the focus of this particular research project. Founded on standard protocols and interfaces, in addition to being facilitated by a number of frameworks and tools, Web Services provide the basis for the development and implementation of business processes distributed over a network, usually the Internet. The basic implementation of Web Services consists of three actors: the service requester, the service provider, and the service broker. The service requester requires services to complete a task and consults the service broker for an appropriate Web Service. The service provider publishes information related to its Web Service with the service broker. When the service broker connects a given service requester with an appropriate service provider by providing the description and location of the Web Service, the communication to accomplish the task can then be directly between the requester and provider. At least two essential components to this architecture are necessary for this to take place: the communication protocol and a descriptive interface.

The communication protocol between the service requester and service provider is SOAP. While in the past SOAP has been considered an acronym to stand for either Simple Object Access Protocol or Service Oriented Architecture Protocol, it is now typically referred to simply as SOAP. SOAP is a protocol which allows the exchange of XML-based messages across computer networks. It provides the basic messaging framework for Web Services. While there are some performance issues perceived to be associated with the use of SOAP, it is a widely accepted standard protocol and is officially the communication standard for Web Services. The descriptive interface which allows the service provider to publish information related to its available services and allows the service broker to
communicate this information to a service requester is the Web Services Description Language (WSDL). WSDL describes Web Services and provides the location that the service can be accessed over the network.

There is a wide variety of tools and frameworks which exist to facilitate the implementation of Web Services. These frameworks can be open source or proprietary and can incorporate a number of different programming platforms including Java, PHP, Python, C, and Microsoft .NET framework supporting a Web Services-based client/server infrastructure. In addition, some frameworks and tools have been extended to provide additional functionality that the basic implementation of Web Services cannot provide. As an example, Ma, Tosic, Esfandiari, and Pagurek (2005) demonstrate how Apache Axis (an open source Web Services framework) can be extended to provide additional functionality for the monitoring of Web Service Offerings.

Figure 3. Orchestration and Choreography

Furthermore, Web services can be arranged, or composed, in such manners as to be able to complete more complex tasks. While a single Web service may only be capable of a single simple
operation, an arrangement of Web services are capable of so much more. Critical to such composition are the aspects of orchestration and choreography. Orchestration refers to an overall executable business process that can be defined through interaction between Web services. Choreography refers to the sequencing of messaging between such executable business processes. The two aspects are related but conceptually different. Figure 3, adapted from Peltz (2003), illustrates the realms the two aspects function within.

These two aspects are enabled through standard languages which facilitate composition activities in a SOA. Web service orchestration is most commonly enabled through the Business Process Execution Language (BPEL). BPEL is a standard which uses XML to describe the control flow between Web services within an executable process and functions on top of the description of the Web services in WSDL. BPEL makes the implementation of SOA easier due to a number of factors. First, it is a standard employed throughout a fair portion of Web service SOAs which helps ensure interoperability. Furthermore, it supports asynchronous message exchanges and provides fault handlers within the business processes (Pasley 2005). Additionally, there are a number of tools which support the utilization of BPEL in a SOA environment. There are commercial BPEL engines and integrated graphic editors which assist developers in modeling their business processes more easily into a SOA framework. Chen, Wassermann, Emmerich, and Foster (2006) identify the Active BPEL engine and the Eclipse BPEL designer as functional open source options in the incorporation of BPEL, while Louridas (2008) illustrates the use of the NetBeans BPEL editor as another example. However, BPEL is not without its limitations. BPEL is not dynamic but must instead be arranged by a developer to structure the business process. Operating at only the syntactic level, it is not yet possible for an autonomous agent to reason and navigate intelligently through WSDL and BPEL to accomplish varied tasks not originally envisioned by the developers. Orchestration can also be represented using WSMO.
the implementation is preliminary. Web service descriptions in WSMO contain the facilities to describe orchestration required for a Web service. Combined within the ontology, orchestration modeling has the capability to operate at a much more semantic level of reasoning than BPEL. Gone and Schade (2008) demonstrate, in the context of geospatial web services, how composition in WSMO can overcome many of the limitations of BPEL.

In terms of choreography, this is often facilitated through the Web Services Choreography Description Language (WS-CDL) standard. WS-CDL is used in conjunction with BPEL to manage the message exchange between business processes and their Web services. Dealing primarily with the order of interaction among Web Services and entire business processes, WS-CDL also makes use of XML. Although an important interface standard for SOAs using Web services, WS-CDL has been criticized for its lack of formal grounding (Yeung 2006). While both BPEL and WS-CDL standards facilitate more advanced use of Web services and enable a structure much closer to true SOA, they are inherently limited due to their operation solely at the syntactic level and are therefore insufficient in terms of enabling reasoning-based automation. As with orchestration, WSMO has the ability to model choreography and the implementation is much more complete than with orchestration. Much like WS-CDL it provides the necessary information required to communicate with a Web service. However, as part of a larger ontology, it also has the ability to support reasoning far beyond what WS-CDL choreography representations are capable of participating in. Stollberg (2005) describes how reasoning tasks and mediation can be facilitated using both the choreography and orchestration descriptions available in WSMO. Composition within WSMO, incorporating aspects of both orchestration and choreography, will be explored in much greater depth in this research.

Overall, the current form of Web Services can only go so far in meeting the needs of an effective and useful SOA. The basic implementation of Web Services generally falls short of what
would be considered a complete SOA. Web Services typically do not support the notion of a contract lease and there are no widely accepted Quality of Service (QoS) specifications (McGovern et al. 2003). Orchestration, choreography, and the mediation between Web services is another challenging aspect to the current practice of SOA using Web services. Furthermore, some implementations of Web Services, especially when using BPEL and WS-CDL, simply do without the role of any service broker when the service requester is created and already has the locational information of the required services. This increases the coupling of the overall system under consideration and is contrary to the principles of SOA. This research recognizes the essential role of a service broker to any automated or semi-automated composition of services and the WSMO family of tools provides such facilities through the WSMX execution environment. More details regarding this functionality is provided in subsequent chapters.

Finally and most importantly, as with SOA in general, there is the inherent difficulty in enabling automated and semi-automated discovery, composition, management, and monitoring of Web Services which are described at simply the syntactic level. Other than providing an interface for the use of a service, WSDL on its own simply does not contain enough information for an automated facility to reason about the services it may need to utilize or make available. BPEL and WS-CDL are suitable for enabling known business processes but are severely limited in their ability to support automated creation of new processes through reasoning. However, the addition of semantic information to Web services and the SOAs that they function within has the potential to enable exactly that. Such services are called Semantic Web Services (SWS).

**SEMANTIC WEB SERVICES.** Semantic Web Services occur at the intersection of two trends in the evolution of the Web: the rapid development of Web Services technologies and the movement towards the Semantic Web itself (Martin and Domingue 2007). The augmentation of WSDL, which
describes Web Services, through annotations available from the Semantic Web, has the potential to facilitate higher automation of service discovery, composition, invocation and monitoring. Among the earliest proposals to create and deploy Semantic Web Services was demonstrated by McIlraith, Son, and Zeng (2001). These authors proposed the markup of Web Services through available Semantic Web markup languages to enable agent technologies and further automate the discovery, execution, composition, and interoperation of Web Services. While acknowledging that there are many possible realizations to achieve this goal, their work was an early proof of concept for SWS.

Markup to create and deploy SWS facilitates a number of key functionalities previously unavailable to Web Services in any consistent manner. Web service discovery refers to the automatic identification and location of Web services which meet a particular requirement of the user, whether human or software agent, and which conform to certain requirements or specifications. Web service execution refers to the ability to automatically execute the selected Web Service and achieve the expected results in accordance with the specifications presented during discovery. Web service composition refers to the ability to automatically compose an application through the use of many Web services to perform more complex tasks than a single Web service can provide. Web service interoperation is related to composition in that it includes aspects of Web service management and monitoring in order to ensure that all Web services are performing as expected and the various actors utilizing the SWSs are achieving the expected results. There are a number of approaches currently being implemented or under study for all the above functionalities. As an example, Charif and Sabouret (2006) detail a number of existing approaches for Semantic Web Service composition.

2.2 THE SEMANTIC WEB

The Semantic Web, sometimes known as Web 3.0, was originally envisioned by Berners-Lee, Hendler, and Lassila (2001) as an extension of the existing World Wide Web which could incorporate
machine-readable content. The vast majority of the current Web is designed to be read and understood by humans. It is, for the most part, not possible for a machine or software agent to freely navigate through the Web and accurately accomplish a task of any significance. Most content on the Web must be viewed by humans and, with the proper context, understood in order to be of any use. A true Semantic Web would add machine-readable structure and encode content in such a manner so that machines, specifically intelligent software agents, could navigate and accomplish tasks by reasoning through the meaningful content of the Web pages.

Such an evolution of the Web requires a great deal of coordination and currently the Semantic Web initiative itself is comprised of a general philosophy, a set of design principles, a number of working groups, and a variety of technologies designed to facilitate this new functionality. The Semantic Web makes use of XML which allows arbitrary structure to be incorporated into Web pages, which can include elements of machine-readable information. This alone, however, does not provide sufficient meaning which would allow software agents to reason and so the Resource Description Framework (RDF) is also utilized by the Semantic Web to express the required meaning. Overall, the Semantic Web is considered as a layer cake with numerous layers of technology contributing to its realization. From the Uniform Resource Indicator (URI) and XML elemental syntax at the low levels of the structure all the way to the ontologies, rules, logics, proofs, and trust structures at the higher levels, it will take a great many enhancements to existing technology to completely fulfill the Semantic Web vision.

As machines become capable of reading, understanding, and intelligently navigating through this Semantic Web, it seems obvious that these same machines would want to avail themselves of the many Web Services available on the Web to perform various tasks. If it makes sense to annotate readable content on the Web with semantic information, it also makes sense to similarly annotate
executable content on the Web as well. Semantically annotating Web Services for the Semantic Web gives rise to an extended form of Web Services already identified earlier as Semantic Web Services. Such services were initially envisioned as a key component to the Semantic Web and make a vast array of capabilities available to the users, both machine and human, of this Semantic Web. However, before such semantics are possible at the machine-level, a key component to the Semantic Web needs to be introduced. Ontologies are needed to constrain and ground the interpretation of the semantic annotation to enable reasoning.

ONTOGRAPHY. In the context of computer science, an ontology is a representation of a set of concepts within a domain and the relationships between those concepts. It is a data model which can be used to reason about objects within a domain. Ontology utilization has always been considered a critical component to the implementation of SWS, and indeed the entire Semantic Web itself. To illustrate this, Sycara et al. (2003) show that capability matching for SWS simply cannot be done properly without the use of ontologies. Without the ability to reference an ontology, reasoning about and within models can only go so far. In order to bridge the reasoning gap within and between systems, both open and closed in nature, the incorporation of ontologies would seem an ideal way to create the connection. For Semantic Web Services, two primary established ontology technologies are OWL-S and WSMO.

OWL-S is based on the Web Ontology Language (OWL) which was originally designed to provide reasoning support in the context of the Semantic Web and to maintain consistency with RDF (Horrocks, Patel-Schneider, van Harmelen 2003). OWL-S itself is a set of markup language constructs specifically devised to support SWS. It can define and describe the properties and capabilities of SWS in a machine-readable and machine-understandable form. Descriptions of Web services in OWL-S contain one or more profiles, a process model, and groundings. The profile is what enables service
discovery while the process model describes how the service can be executed. The grounding definitions are used to govern the messages used in communicating with the Web service. OWL currently has three variants: OWL Lite, OWL DL, and OWL Full. OWL Lite is the least expressive while OWL Full is the most expressive. OWL-S is based on OWL DL which, based on Descriptive Logics (DL), is the most expressive variant which retains complete decidability. As OWL Full cannot guarantee decidability, it would be unsuitable for reasoning and therefore entirely unsuitable for a Web service description ontology. The wealth of existing research in DL supports the reasoning tools and methodologies currently available for OWL-S. The well-established OWL-S is also the most prominent of the SWS description efforts (Balzer, Liebig, and Wagner 2004).

Although a more recent innovation, the Web Service Modeling Ontology (WSMO) was also designed to semantically describe all relevant aspects of SWS (Roman, Keller, Lausen, de Bruijn, Lara, Stollberg, Polleres, Feier, Bussler, Fensel 2005). WSMO consists of four top-level, or core, elements: Ontologies, Web Services, Goals, and Mediators as shown in Figure 4.

Figure 4. The Top Level Elements of WSMO

Ontologies within WSMO consist of concepts, relations between the concepts, instances, and axioms. Services (Web services, to be precise) are described by non-functional properties, imported ontologies, any mediators used, the capability of the existing service itself, and any interfaces. The interfaces correspond to the orchestration and choreography aspects described earlier in this section. A
Goal can be considered similar to a Service. While a Service describes a Web service which exists, a Goal describes a Web services which is desired. It is the Mediators which handle heterogeneity between the elements and are capable of resolving mismatches within and between the groups. There are four types of mediators: OO Mediators, GG Mediators, WG Mediators, and WW Mediators. OO Mediators resolve differences between ontologies. GG Mediators resolve differences between Goals. WG Mediators resolve differences between Web Services and Goals. WW Mediators resolve differences between Web Services. Resolving terminology and semantic mismatches, Mediators are key elements in the WSMO framework. As WSMO is a relatively new technology, it may still require further revision and correction (Wang, Gibbins, Payne, Saleh, and Sun 2007) before it is as established a standard as OWL-S. However, it is an important emerging technology which has a great deal to offer SOA. The two ontologies differ in many respects. However, an in depth comparison of these two primary SWS ontologies is deferred to the Methodology section following as it is at that point where a decision between the two for the purposes of this research is evaluated. Nonetheless, both are important standards for describing SWS in a SOA context.

2.3 SOFTWARE PRODUCT LINES

We now move from the very open world of the Semantic Web and its associated Semantic Web Services to an environment which is considerably much more closed in nature and more associated with software development and production strictly within organizations. The concept of a Software Product Line (SPL) harnesses the principles of industrialization and automation in order to make the development of software more efficient in addition to the resulting artifacts being of higher quality. SPL practices are typically incorporated within software development organizations and are focused on the planned structure and intentional reuse of components to achieve a number of benefits.

SPL practices involve following a well-defined methodology to precisely represent the domain
the software must function within. The maintenance of variation points in order to facilitate the exploitation of commonality and the management of variability among software features is a key principle of SPL. SPL is significantly more than just a simple reuse scheme but is an overall paradigm for the development of software.

The benefits of incorporating SPL practices are well known. Schmid and Verlage (2002) describe the positive economic impacts of SPL adoption. SPL practices have been shown to decrease the cost of developing software, reduce the time-to-market for software products, and increase the overall quality of software artifacts within adopting organizations. Software is increasingly being seen as a commodity and therefore organizations must consider how software can be made cheaper, quicker, and of higher quality in order to remain competitive. Despite initial barriers to adopting SPL practices that some organizations have experienced, new methods continue to be devised and developed to enable SPL adoption into mainstream software development practices (Krueger 2006).

SPL practices exist in numerous domains and the literature contains many examples of successful adoption. Kang, Lee, and Donohoe (2002) show how feature-oriented SPL practices can be incorporated into the home integration system market. Trask, Paniscotti, Roman, and Bhanot (2006) demonstrate how SPL practices can be combined with other advanced software development practices, in particular Model-Driven Engineering, to create an advanced product line for the production of radio components and related applications. White, Schmidt, Wechner, and Nechypurenko (2007) automate SPL practices in order to derive software variants for mobile devices. Even in high-performance mission-critical systems, we see SPL practices being adopted as Lutz and Gannod (2003) demonstrate SPL applicability with regards to interferometry devices used with spaceborne telescopes.

While SPL implementations are typically internal closed systems, there has been an increased interest in moving SPL beyond the internal organization. Expanding SPL practices into other
organizations, technologies, infrastructures, and uses has the potential to present many benefits. While there are a number of gaps which need to be filled before this is widely possible, SPL practices are already being modified in anticipation of this. For example, Taulavuori, Niemela, and Kallio (2004) show how the practices of component documentation within an SPL infrastructure are being standardized to better enable automation and integration between other systems. It should be clear at this point that a primary component in any such integration would be the utilization of ontologies to link the closed-world of SPL practices with the open-world of the Semantic Web as described above. However, it is first necessary to further describe a specific representation commonly used in SPL practices which would better prepare an SPL to participate in such a linkage. This is the feature model.

FEATURE MODEL. SPL implementations are typically feature-based as features are a logical point of variation for any given group of software products. Therefore, the feature model is an extremely useful method for modeling the commonality and variability within an SPL. The initial concept of feature models for software reuse was formalized as early as 1998 by Kang, Kim, Lee, Kim, Shin, and Huh (1998). Their Feature Oriented Reuse Method (FORM) utilized a feature model to define the decision space for application development. The feature model itself modeled the domain of the software and the creation and maintenance of that model is known as domain engineering. The creation of any specific configuration from the feature model is known generally as application engineering.

A feature model consists of a feature diagram and associated information, including rationale, constraints, and dependency rules. The types of features modeled in the feature diagram include:

- Mandatory features
- Optional features
- Alternative features
- Or features
- Optional Alternative features
Optional Or features

However, these are currently only the features that are represented in a strictly graphical manner in the model. There are other relationships and properties which need to be represented in the feature model including the Requires relation, which means that the selection of one or more features necessitates the selection of one or more other features, and the Excludes relation, which means that the selection of one or more features negates the selection of one or more other features. In addition to these non-graphic relations, a feature model can contain any number of annotations to further represent certain features or more complicated relationships.

The feature model itself for any given SPL represents all possible valid configurations of that SPL. A product configuration would represent simply one complete navigation through the decision space of the feature model and a resulting software product. Feature diagrams can be generated from a formula and can also be computed into a formula (Czarnecki and Wasowski 2007), though it is important to note that there is the issue of semantic equivalence where a given formula could lead to the creation of feature models very different in appearance yet representing an equivalent decision space (Sun, Li, Zhang, Wang 2005). However, these qualities of feature models do allow for advanced logics and reasoning to be incorporated in their use.

The ability to support reasoning is an important characteristic for such a model. In order to truly automate the facilities constructed to support SPL, logical reasoning must be supported. Janota and Kiniry (2007) demonstrate how higher-order logic can be utilized to reason about and within feature models and their resulting configurations. To further support reasoning abilities, Benavides, Trinidad, and Ruiz-Cortes (2005) show how feature models can be extended to better support the reasoning functions which can further automate SPL implementations. As SPL solutions increase in both size and scope, the ability to automate such tasks becomes critical. However, there are limits to the
expressiveness of feature models as they current exist (Czarnecki and Wasowski 2007) and this would severely limit the extent to which reason-based automation could be incorporated. Fortunately, there exist ontology technologies which can be employed in this instance to make more advanced reasoning and therefore greater automation possible.

2.4 INTEGRATION OF SOA AND SPL

While the literature concerning this particular research direction is relatively sparse due to the novelty of the technologies involved, there is nonetheless a wealth of relevant research which this project can draw upon in order to better formulate its approach. The synergies which this research takes advantage of require an in depth review of a wide variety of related topics and approaches in order to facilitate the connections between the SWS and SPL fields of study. It must, of course, first be established that integration between SOA and SPL makes sense conceptually. There is a body of research which exists that supports this approach. Helferich, Herzwurm, Jesse, and Mikusz (2006) demonstrate how SPL and SOA do not have to necessarily be considered mutually exclusive concepts. The authors propose that a hybrid model of software development may emerge where SPL components and SOA services co-exist within systems. There are shortcomings apparent in each approach which can be complemented by the other. Trujillo, Kastner, and Apel (2007) pursue this idea further and demonstrate the use of SOA for the integration of several SPLs, in a scenario they describe as a service-oriented product line (SOPL). In their work, the reuse of services through a SPL can satisfy diverse variability requirements with little human intervention. To be able to treat services as components of a SPL, the granularity of these services must be considered. The work of Lee, Muthig, Naab, Kim, and Park (2008) defines reusable software assets among services, distinguishing between atomic services, the much richer molecular services (composed of atomic services) which offer usable computational functionality, and the orchestrating services which allow definition of workflows. This is a key issue in
representing a SOA as a SPL. However, in order to support the level of reasoning required to create a synergy between SWS and SPL, we need to delve into greater detail than illustrated in the above works. The reasoning support for verifying and deriving product configurations from an SOA defined by feature models would need to be facilitated by ontologies. The literature which explores the use of ontologies with SWS and also the use of ontologies with feature models is now considered.

Considering the interaction between SWS and ontologies, Sycara et al. (2003) clearly show that capability matching cannot be done for SWS without the use of ontologies. Their work incorporates the creation and deployment of ontologies using DAML-S (a predecessor to OWL-S) to develop well-defined semantics that can support automation and dynamism in Web service discovery, composition, invocation, and management. Noting that a great deal more research still needs to be done in order to achieve automated interoperation between Web services, the authors nonetheless show that ontologies can be used to support these functions (and in fact, it cannot be done without them) and that true semantic interoperability is possible between such services. However, although high-quality ontologies are essential for SWS, Sabou, Wroe, Goble, and Stuckenschmidt (2005) demonstrate that ontology creation for SWS is a difficult and expensive endeavour. The authors propose that ontology creation should be automated as much as possible to alleviate this expense and therefore it is necessary to utilize tools which can learn an ontology from a given collection of services and their descriptions (i.e. WSDL). Interestingly, the authors acknowledge other sources that could be used to assist in the learning of Web service ontologies. These include source code, textual documentation, or UML diagrams. Although beyond the scope of their work, it is logical to suggest that a reasonable extension of that list of potential other sources would be any available feature models and resulting configurations if the SWS were organized in a SOA and modeled as a feature model.

The work of Brambilla, Ceri, Facca, Celino, Cerizza, and Cefriel (2007) brings SWS much
closer to the ontologies which can support them. The authors work with the emerging WSMO standard to design and develop SWS. Providing an excellent case study in the use of the WSMO conceptual model and related tools, the authors are able to describe SWS in such a manner as to enable reasoning over the ontologies which support the SWS. Balancing expressiveness and tractability, the research demonstrates how significant portions of the WSMO-compliant descriptions of the SWS can actually be derived in a semi-automated fashion from certain artifacts of design specification. The percent of automation for deriving composition descriptions in WSML is actually quite high which should be expected considering they are using relatively expressive Business Process Modeling Notation (BPMN) artifacts as part of their input. Their work serves as a benchmark for the automation which is possible to achieve in deriving composition descriptions (primarily choreography but also orchestration) from process modeling artifacts. Stepping back from WSMO into the OWL-S regime, Lee, Kim, Lee, and Lee (2007) combine the use of UML and OWL-S to support automation of service discovery, invocation, and composition within SWS. They propose a new service description method while developing a prototype to show service discovery and evolution at runtime.

The ability to utilize graphical constructs available through a language like UML would seem to be more and more necessary when considering the increasingly complex nature of a SOA modeled through SWS. Following this practice further, the reader can consider how a feature model could participate in the same relationship. Gašević, Djuric, Devedzic, and Damjanovic (2004) demonstrate how OWL ontologies can be derived from UML to support automated development of ontologies from models created through UML. Lautenbacher and Bauer (2007) take this a step further in order to create a metamodel and UML-profile for all SWS standards. The authors detail a survey of existing standards and then show that the proposed metamodel can be applied and transformed into all the listed standards (including WSMO and OWL-S). This has the potential to bring the semantic description of web
In the realm of feature models, ontology utilization is also being realized. Relevant literature pertaining to the general reasoning about and within feature models has already been described in an earlier section so this will focus specifically on the reasoning connection between feature models and ontologies. Czarnecki, Kim, and Kallberg (2006) demonstrate how feature models are best considered as views on ontologies and that a feature model itself represents a family of ontology constraints. Acknowledging the many-to-many association between elements in feature models and elements in ontologies, the authors nonetheless propose mappings to give semantics to feature models which can enable a number of advances in their use. Wang, Li, Sun, Zhang, and Pan (2006) move beyond the consideration of feature models as views on ontologies to where feature models are verified using ontologies—specifically OWL-based ontologies. This furthers the ability to reason within feature models. The authors propose methods for transforming feature models into OWL ontologies and utilizing reasoning facilities to perform automated analysis over the resulting OWL representation of the feature model. The authors do note the limited expressiveness of OWL for this purpose and indicate that a more expressive language may be needed to support more mature reasoning tools. They conclude that the Semantic Web can play an important and synergistic role in the domain engineering activities associated with SPL. The research of Dodero, Sanchez-Alonso, and Frosch-Wilke (2007) manages some initial mappings between ontologies in WSMO and feature models in their work in the development of competence development programs in the education field. In their mappings, the goals within WSMO are associated with the feature configurations from the feature model.

Peng, Zhao, Xue, and Wu (2006) explore ontology-based feature modeling. Using an ontology as the basis for feature modeling, feature models are converted into OWL and then validated through ontology reasoning. This is shown to support both domain model validation as well as providing
guidance to application engineering efforts. Interestingly, the authors even suggest that it can be applicable in the support of run-time reasoning. Their research also demonstrates an ontology-based feature modeling tool. Lee, Kim, Song, and Baik (2007) also pursue the potential connections between ontologies and feature models through the creation of a transformation between feature models and a feature-ontology based on “semantic similarity mapping.” This is also done using OWL and demonstrates a tool created for automation of semantic feature comparison and mapping. Both of these research works support the concept of transformation between ontologies and feature models though primarily in one direction, from the feature model to the ontology.

The work of Kim (2006) takes these concepts further and also considers such transformations in both directions. Working under the premise that the less expressive feature model is merely a view onto the more expressive ontology, transformations from the ontology to a feature model are termed “view projection” while transformations from feature models to ontologies are termed “view integration.” In view projection, features are derived from a mature ontology and the ontology also supports decisions regarding the location of the features in the feature model hierarchy. Transformation goes beyond mere syntactic correspondence but also involves semantic mapping. In view integration, one or many feature models are used to create an ontology (or augment an existing ontology). View integration is also suggested as being able to address a limitation of feature modeling, that of composition. In both directions however, there is a significant amount of manual intervention required and there appears to be ample opportunity for further work which could be done to increase the level of automation achievable.

The majority of these transformations detailed in the literature work solely with OWL-based ontologies and are limited in terms of the reasoning that they facilitate. The ability to transform the business process information contained within feature models and their product configurations directly
into ontology-based SWS descriptions to support greater reasoning is not adequately demonstrated in any literature found during this review. The developer conducting feature modeling activities with a SOA has little way of knowing whether resulting product configurations are optimal or even valid. Apparently, there does exist a gap in the current state-of-the-art which needs to be filled in order to support efficient implementation of such mappings between the SOA and SPL worlds, in addition to reflecting the growing popularity of WSMO compared to OWL-S and the importance that feature models have in fulfilling the requirements, and delivering the benefits, of SPL architectures.

2.5 MODEL DRIVEN DEVELOPMENT

There is a significant role for Model Driven Development (MDD) to play in this research, both from a conceptual viewpoint and also from utilization of available tools and technologies incorporating an MDD approach. MDD is a development paradigm which raises the level of abstraction to that of the model. Known to increase productivity and reduce time-to-market, in addition to being associated with the early (and therefore less costly) detection of errors, MDD is an important emerging paradigm (Balasubramanian, Gokhal, Karsai, Sztipanovits, Neema 2006). In addition to working at a higher level of abstraction which allows domain experts to directly contribute to development, a key premise behind MDD is the automatic generation of code from the model. MDD is enabled by various automation technologies and tools, as well as a number of the same standards we see in the Web services domain such as XML and SOAP (Selic 2003). In addition to making use of tools, technologies, and standards which implement aspects of MDD, this research considers that both WSMO ontologies and feature models are, in fact, models themselves. Therefore MDD concepts will be able to form part of the approach. This is especially true considering a requirement to perform model transformations between feature models and WSMO. The well-defined practice in MDD of metamodelling to govern the conduct of model transformation will certainly assist in the required transformation activities in this
work (Sendall and Kozaczynski 2003). As the emerging paradigm of MDD is becoming more relevant and utilized throughout the field it is important to take advantage of its functionalities to both realize efficiencies and to keep the work relevant to industry practices and trends. Model transformation, the act of producing one model from another model of the same system, is a primary activity of MDD.

Milanovic, Gašević, Giurca, Wagner, Lukichev, and Devedzic (2007) demonstrate model transformation methodologies and tools used to bridge between the concrete syntax and abstract syntax of Web rule languages (SWRL, R2ML, and OCL) which are used to reason over Semantic Web ontologies. With models specified by modeling languages, which are themselves defined by metamodels, the authors are able to create a model transformation enforcing valid source and target models. Metamodeling is demonstrated as a key component to the solution. The authors’ evaluation of the methodologies and tools utilized in this work to define the mappings and execute the transformation will contribute to the discussion of tools and methodologies selected in the following chapter. In another example using models relevant to this research, Moran and Mocan (2005) demonstrate bidirectional mappings for model transformations between XML Schema, specifically Web services descriptions in WSDL, and WSMO Ontologies expressed in WSML. Both works deal with the model transformation processes at the semantic level, above and beyond merely handling the different syntax of the models. This type of approach is a key element in this research.
CHAPTER 3
METHODOLOGY

Our approach will treat an architecture as one model with two different formalisms, one reflecting the service oriented architecture (SOA) context and one reflecting the software product line (SPL) context. However, in order to provide reasoning support for modeling a SOA within an SPL context through the use of ontology, we need to map between the SOA and SPL representations. To complement the process identified by Trujillo et al. (2007) when they suggest that "when the SOA application itself turns into a product line, a new scenario emerges," this work requires the intentional transformation of the models first representing the SOA as an SPL and then for subsequent product configurations and model revisions within the SPL back out into the SOA environment.

Figure 5. General Methodology Outline
The transformation is a primary deliverable of this research but conducting the transformation alone would not be a comprehensive solution without a thorough evaluation of the resulting ability to support reasoning and offer improved methods for software product line practices. To evaluate the transformation and the reasoning it facilitates, we use a set of input services and a prototype framework which will support composition, reasoning, verification, and testing to assess and improve the transformation and the new functionality with the feature model view. This general process of the work follows that shown in Figure 5.

In describing the overall research methods and approaches we begin by briefly outlining the selected model to represent the services of an existing SOA as features in a SPL approach, the Feature Model. We then describe the target ontology-based model, WSMO, which will allow valuable reasoning to be done on the resulting product configuration. The approach and methodology for transformation required between these two models is an important section following and this is a key enabling concept in this research. The reasoning which is then made available to the feature model view, but supported by the ontology, is then subjected to exploration, verification, and analysis. The development of a prototype and testing framework is then described generally.

3.1 FEATURE MODELING OF SERVICE ORIENTED ARCHITECTURES

In terms of the type of SPL implementation which is used in this project, the feature model appears to be the most appropriate. This is based on its wide usage in the SPL community, the tool support which currently exists, and also the wide range of literature available regarding reasoning about and within feature models that this research can draw upon for support. There are various approaches to feature model notation but this research has selected the one proposed by Czarnecki et al. (2006) partially because of its popularity within the field and available tool support but also because of its comprehensiveness in terms of clearly representing diverse relationships among features. It is
important to note that this research incorporates both the metamodel of feature models as described by those authors but also the rendering. The metamodel is the most important aspect while the rendering is more a matter of convenience and compatibility with the tools available. Realistically, other renderings could be utilized provided the metamodel was consistent.

Lee et al. (2008) demonstrate a feature-oriented product line approach for SOA which guides developers through the organization of services into orchestrating services and molecular services within a feature model, addressing the key issue of granularity. However, while it may be straightforward to implement a methodology where, in a SOA represented in a feature model, the lowest-level features would represent actual molecular services, there needs to be a consistent structure enforced so that something resembling an actual composition can be derived from any configuration from the model. Without this, a transformation between the two formalisms has no way of inferring any business process information from the feature model, which is essential in creating an executable orchestrating service description to test through ontology-based reasoning. Product configurations from a standard feature model do not inherently indicate any information regarding the business process that could potentially be involved. The additional step which allows feature models to represent service composition is the introduction of the conceptual 'ordering' of services to the practice of feature modeling. Although the order of features on the same level within a feature model typically has no meaning, in this case the developer must follow a certain style to ensure that the feature model is reflective of the general order of the possible flow of the business process. This order will be reflected in the transition rules description in WSMO, which is conceptually based on the Abstract State Machine (ASM). The style is adapted from Montero, Pena, and Ruiz-Cortes (2008) and the details are illustrated in the detailed design following this chapter.

3.2 COMPOSITION IN THE WEB SERVICE MODELING ONTOLOGY
An essential element to this work involves the description of an appropriate SOA design that this process can support and enhance. The choice of the particular ontology framework to support the SWS aspects of this research is difficult. Two contenders are OWL-S and WSMO. This work is electing to pursue the WSMO framework (http://www.wsmo.org/) for a number of reasons which are made clear in the relevant literature. Shafiq, Moran, Cimpian, Mocan, Zaremba, and Fensel (2007) conduct an extensive investigation and comparison between the OWL-S and WSMO frameworks, with particular focus on the available tools for each. The comparison looked at a number of perspectives including: service discovery, data storage and management, mediation, execution management, choreography and orchestration, programmatic access support, end user interaction support, grounding, reasoning support, security issues and fault tolerance. For the purposes of this research, of primary interest are those comparisons related to service discovery, reasoning, orchestration, choreography, and, for the purposes of prototype implementation, end user interaction support. The overall comparison results showed that WSMO compared favourably against OWL-S in the most important categories. Lautenbacker and Bauer (2007) also show in their survey of existing standards that WSMO is more expressive than OWL-S as it incorporates descriptive logic (as does OWL-S) but also first-order logic as well. Balzer, Liebig, and Wagner (2004) identify a number of “pitfalls” associated with OWL-S and the limitations associated with the declarative foundations it inherits from its parent language OWL, in addition to deficiencies in its conceptual model. WSMO also has a relatively mature semantic execution environment, the Web Service Execution Environment (WSMX, http://www.wsmx.org), which can reason against semantic descriptions of Web services beyond reasoning only against their associated domain and process ontologies. Although acknowledging that WSMO is a relatively new technology and has not necessarily been subject to the same use and rigorous experimentation as OWL-S, its benefits are clearly worth pursuing in this particular research. Other options which were
originally under consideration were WSDL-S and SAWSDL both of which add semantic annotation to WSDL files but were judged not to be as comprehensive of a framework as needed to employ in this particular research. Therefore, as the WSMO framework has been selected, transformation and reasoning with any other ontology frameworks associated with SOA are beyond the scope of this research. However, the reader may be able to draw parallels when considering those other technologies.

It is important to note that the WSMO framework includes a number of components. WSMO itself refers to the conceptual model for describing the core elements (Web services, ontologies, goals, and mediators) while the Web Services Execution Environment (WSMX) is the execution environment for the discovery, invocation, and interoperation of services. The formal language which supports the writing, storage, and communication of the related descriptions is the Web Service Modeling Language (WSML). The Web Service Modeling Toolkit (WSMT) also exists to support this framework and is utilized for editing of .wsml files where manual intervention is required.

With the feature model representing available services, there needs to be a way to represent a valid product configuration from the feature model to WSMO. When considering that the features of a feature model representing a SOA are in fact actual services, it would logically suggest that composition is an appropriate means by which to represent a configuration in WSMO. A Web Service element in WSMO is composed of a capability and an interface, in addition to non-functional properties. The capability consists of shared variables, preconditions, assumptions, postconditions, and effects. Capability descriptions are essential in the Web Service element as that is the most detailed information that current discovery engines implementing the WSMO specification act on. Lightweight-discovery is based on postconditions only while heavyweight-discovery can use more information from the capability description. In any case, the capability description must describe the service sufficiently so that it can be discovered.
The interface consists of a choreography and an orchestration, which would comprise the composition of Web Services into certain business processes. Orchestration refers to an overall executable business process that can be defined through interaction between Web services. Choreography refers to the sequencing of messaging between such executable business processes. Conceptually, while a product configuration from a feature model would refer to one or more choreographies; it is itself more of an orchestration from that perspective. Both orchestration and choreography descriptions implement the ASM and are comprised of a state signature and transition rules. The state is typically in the form of an ontology which provides both the vocabulary for the transition rules as well as defining the set of instances that change state. In our transformation process, this state ontology is derived and extended by the elements of the product configuration. In addition to the base information imported from domain and process ontologies, the state also requires important properties *mode* and *grounding*. The state signature contains all the mode containers which define the state of the ASM. They can include *static, in, out, shared, or controlled* mode containers. These mode containers can also contain the grounding information (in the form of an IRI) indicating where these containers can be accessed on the Web. The transition rules specify how the ASM changes state from one transition to another, utilizing the mode containers and control flow constructs including *if then* conditional statements and *for all* loops. The guarded transitions are a set of rules which are triggered as certain conditions are met. It is these rules which determine which Web Services are composed to meet the business processes requested through the product configuration. The transition rules of the choreography description specify how information is exchanged between the client and the service. The transition rules of the orchestration description specify how other molecular services are engaged in order to accomplish the task for the orchestrating service.

Unfortunately, currently in WSMO only the choreography standard is fully specified and
implemented. The WSML language specification for orchestration is not finalized and, furthermore, existing tool support does not fully take into account the possibilities provided by orchestration. However, although the WSMO Orchestration model is still under some development (Scicluna, Polleres, Roman, Fensel 2006), it is sufficiently developed to consider in relation to this work. This issue was also noted in the work of Brambilla et al. (2007), but nonetheless they were able to assume a reasonable WSMO orchestration representation in their process. However, as choreography in WSMO is a much more advanced implementation than orchestration, the utilization of choreography beyond its intended use has been demonstrated often in the literature and will be necessary in this work. To some extent, the same effect of orchestration can be achieved through the choreography description and this will be shown in this work.

3.3 TRANSFORMATION BETWEEN SOA AND SPL VIEWS

This research required a mechanism for transforming between the two formalisms of the model, from the feature model product configurations which need to be reflected in WSMO. Generally, transformations can be challenging to develop, understand, and maintain. In order to create such a transformation, the developer must understand fully the syntax and semantics of both the source and target formalisms. Fortunately there are approaches which are available to assist with this. Foremost among these is Model Driven Development (MDD). This process follows MDD principles, allowing the developer to model features of a SOA in a SPL formalism with substantial portions of semantic code requirements automatically generated. This ensures that output is relatively easy to generate and minimizes the many potential errors which can occur through manual coding of semantic Web service descriptions.

As this research is looking at the SOA as a model (or more appropriately one formalism of the model), it is important to look at how the SOA can be represented in a consistent manner to facilitate
the transformation and later reasoning. The best approach is that of the metamodel. In addition to providing the most appropriate view into the syntax and semantics of WSMO, the use of the available metamodel will ensure relevance of the findings of this research to other as yet undeveloped uses. The metamodel for WSMO is sufficiently introduced in Wang et al. (2007) and Roman et al. (2006). Keeping the focus of the transformation and later verification and reasoning efforts on the metamodel approach should also make the improvement and, if necessary, evolution of the tools easier. This is especially important considering that WSMO is a relatively new technology and the representation and interpretation of its semantics have been known to vary among different uses. It is also expected that WSMO may evolve and for this research to maintain relevance during evolving specifications, the metamodel approach offers the most practical solution. A metamodel for WSMO Ontology, Web Service, and Goal elements (Mediator elements were not implemented) suitable for incorporation into the transformation environment was developed in the KM3 format. It is outlined further in the detailed design. Furthermore, given that WSMO is most often used in its textual concrete syntax, WSML, it was also necessary to develop a parser and lexer for this language to allow the transformation to extract the actual WSML descriptions.

As with the WSMO formalism, it is important to consider the contribution that the metamodel of the feature model can make to ensure the transformation is comprehensive and easier to maintain as standards evolve. The metamodel for feature models is described within Benavides, Trinidad, and Ruiz-Cortes (2005) and further in Benavides, Trujillo, and Trinidad (2005). Using the feature model as the source SPL formalism for the transformation makes the process partially dependent on the chosen tool (described later) in that it is important to use a format which is implemented in a given graphical feature model environment. Fortunately the metamodel for feature model is well-defined and the selected feature model environment also is capable of importing and exploring appropriately formatted
XML. Even more fortunate is that the metamodel used by the selected tool support for feature modeling is Ecore-based, greatly facilitating the transformation.

Focusing on the higher level of abstraction, and separating the model and the code, allows this transformation to be easier to comprehend and maintain, in addition to being more relevant for other potential uses. It also helps ensure that the transformation is comprehensive when the input SOAs in this research do not necessarily lead to the exercise of all aspects of the model. There is, of course, some work in moving the data from the two metamodels to the working formats but at least the developer can be confident that the flow between the two metamodels is preserved should working formats evolve or change.

This research has developed a model transformation between the feature model and the WSMO representation of a service oriented architecture. Using the terminology utilized by Kim (2006), and following a process conceptually similar, the transformation from feature model to ontology is considered a more advanced "view integration". These concepts and their associated terminology will be preserved in this research. The model transformation itself is composed of mappings (between the two metamodels) and transformations (between the two different concrete syntaxes). This is illustrated in Figure 6.

Figure 6. Mappings and Transformations
Figure 7 shows a simplified metamodel of the feature model, adapted from Kim and Czarnecki (2005).

Figure 7. Feature Model Metamodel

Figure 8 illustrates an abbreviated metamodel of the WSMO Ontology, focusing specifically on the elements which have representation in the conceptual mappings with the feature model.

Figure 8. Abbreviated WSMO Metamodel
Conceptually, the transformation considers the order of the services selected in the product configuration and uses that as the base to structure the resulting choreography and orchestration descriptions. The capability description considers that the first selected molecular service's preconditions will likely be preconditions of the orchestrating service while the final selected molecular service's postcondition is logically a postcondition of the overall orchestrating service. This is fairly logical though in practice requires some manual intervention to ensure a completely executable service description. The state signature description is basically a composite of the input and output mode containers of the individual molecular services. The transition rules consider the order of the services within the product configuration and represent that back out in the ontologized abstract state machine (OASM) which WSMX can interpret for the purposes of managing the various transitions. Within these conceptual heuristics there are smaller, yet important, mappings which bring the transformation result closer to a discoverable and executable Web service description. These include certain Non Functional Properties (NFP) as well as namespace and WSML variant information required for valid .wsml files.

In summary, the input to the transformation is a valid .fmp file from the Feature Plugin, which must contain a product configuration. Upon applying the above mentioned mappings, the output is a .wsml file which can serve as input to a testing framework utilizing the WSMX execution environment. The complete transformation is done in two parts, with the main utility conducting the transformation between the Ecore-compatible .fmp file to an interim WSMO Ecore-compatible XML file. Given that this WSMO XML is not really supported by any available tools, there is a further transformation to translate this model into the textual concrete syntax of the WSML languages that the family of tools supports. The mappings are completely defined in the detailed design illustrated in Chapter 6.

From this point, we have a set of mappings which can harness the reasoning capabilities
available through the ontology-related tools. The ability to validate and query the resulting composition against the ontologies can provide useful information to feature modelers. However, some further investigation is necessary to determine the level of manual intervention required to maintain a valid composition from the transformation. These conceptual mappings, though useful, leave additional information between the models unrepresented. Product configurations from feature model often contain other information, including attribute values, which may be able to be transformed into WSMO to enable further reasoning. These mappings need to be extended to incorporate any attributes and attribute values of the feature model. There are of course elements in WSMO which do not map at all to the less expressive feature model and this is to be expected and compensated for through manual intervention. It is expected that through the experience of reasoning within the execution environment, the conceptual mappings are refined and improved to develop more accurate mappings with the least amount of manual intervention. It is acknowledged that complete removal of manual intervention from the process is not possible in the general sense, and likely not desirable for many specific applications. Detailed instances of manual intervention are illustrated in the detailed design in Chapter 6.

3.4 VERIFICATION, REASONING, AND ANALYSIS

The main goal of this research is to use a SWS description language, in this case WSMO, to encode product configurations that can be verified and validated. Basically this requires the transformation which is described in the section above. Although we are acquiring the overall domain model from the SOA and are able to encode product configurations in such a manner that they can be reflected in the ontology, in order to ensure that these feature model products are valid, discoverable, composable, with the ability to mediate with other components within the SOA, the existing ontology must be harnessed to support the required reasoning. Within Chapter 2 a number of examples where simple reasoning is supported in feature models without ontologies were discussed. Even without an
ontology, basic questions such as whether the feature model is valid, how many products can be derived from the feature model, and the levels of variability and commonality can be answered. However, this is fairly limited in terms of utilizing feature model practices to support SOA development. There are much greater requirements for reasoning and WSMO provides the basis for these. The feature model is a convenient intuitive modeling environment for SPL developers to model within, but it is also a less expressive notation and its representation of relationships and information is not as complete as an ontology. However, considering that the feature model is a view that developers would like to model within, we need to evaluate what reasoning can be supported by an ontology in this environment and also what guidance an ontology can provide while feature modeling. Links between the feature model and WSMO have already been established and now some value must be extracted. Synergies are available provided we are able to harness the reasoning engine in the ontology environment and project its information to the feature model view.

In terms of encoding product configurations in SWS languages, the developers need to be able to compute that the configurations are valid, that they map to actual services, that they can be composed in an appropriate manner to support business processes and that when they are composed the services communicate properly with each other. The ontology and the SWS descriptions can provide access to this information when reasoning is employed from the feature model against the ontology and other SWS information from the WSMO framework. It is possible to do some initial verification utilizing such methods as critical pair analysis as done in Jayaraman, Whittle, Elkhodary, and Gomaa (2007). However, some modifications should be made to this process as there should be no need to transform the feature model into UML when we already have the more expressive WSMO environment at our disposal. Further verification support for both the domain model and product configurations could follow that work done by Wang et al. (2007). However, as WSMO is more expressive than OWL,
and we already have ontologies and molecular SWS descriptions in WSML, we are able to enhance the value for the developer to enable support for reasoning beyond this. The developer should be able to use the prototype and testing framework to consume the details of an SOA represented by valid WSMO descriptions, with the feature model environment and then refine that model and create new product configurations. Product configurations themselves require further validation against the SOA model and its ontologies. The heuristics of mapping a product configuration to a SWS language support the verification of the product configuration against the SOA and its ontology.

In order to evaluate and verify the solution, a number of criteria will be employed. They relate to testing the accuracy of the mapping possible between the two formalisms in addition to the degree of reasoning that they facilitate. They are as follows:

- What is the level of automation which can be achieved?
- What is the precision of feature discovery?
- What forms of guidance can be provided from the ontology while feature modeling?

The criteria do not lend themselves well to quantitative evaluation in the strictest sense, although the level of automation will lead towards a generalized percentage of manual intervention that is required depending on the levels of abstraction involved between the models. Some measures of performance such as duration of process execution are also considered. The precision of feature discovery may also lend itself to some initial patterns that may lead to quantification in future research. Evaluating the guidance which can be provided is, for the most part, qualitative but this will nonetheless prove a useful benchmark in a comparison with other state-of-the-art solutions which are related to this work.

Stollberg (2004) identifies key reasoning tasks which can be associated with WSMO composition information. For choreography, the primary reasoning tasks are related to service compatibility. The subtasks which need to be addressed are whether the services use compatible ontologies, whether the information to be exchanged is compatible, and whether the communications
protocol between the two services is sound. Where there are deficiencies, the requirements for mediation must be identified. For orchestration, the primary reasoning tasks are to address whether the orchestration is valid and, specifically, whether it is compliant with the choreography. While conceptually these reasoning tasks are straightforward, the use of components with reasoning capabilities within the WSMX execution environment are required to satisfy these information requirements.

3.5 PROTOTYPING AND TESTING FRAMEWORK

There are a number of tools which this research employs to carry out all required tasks. In order to efficiently evaluate and iteratively improve the transformation and reasoning functionality developed by this research, it is necessary to have developed a prototype and testing framework. Supporting the MDD approach, there should be little transformation code which is dependent on the content of the input model—only the specifications of its metamodel. This enables automation, iterative improvement, and the running of as many tests, with as many different input SOA, as required to perform a rigorous validation of the implementation. The internal details of any business logic should be decoupled from the implementation. Although, as with any prototype it is subject to potential abandonment when it outlives its original purpose, there is the potential for some elements of this prototyping and testing framework to contribute to a larger body of work. As such, great care is taken to ensure proper software development practices are used.

A number of software development approaches are available for prototype creation, test case Web services development, as well as the creation of any tools which are necessary to enable this research. From both the perspective of utilizing existing source code and the creation of any new source code, the open source paradigm is used for a number of reasons. There is a wealth of open source tools and source code modules available to support this research. The utilization of these assets
can have a profound effect on the efficiency and cost effectiveness of this research, specifically when developing prototypes. Provided an infrastructure for supporting diverse open source components is employed in any software development activities related to this project, such as that described by Obrenovic and Gašević (2007), the complexity which may exist in the integration of these diverse assets can be managed appropriately. Furthermore, in the interests of supporting other research and contributing to the larger field of study, any useful tools or modules created through this research are made widely available through an open source license.

In terms of specific software development methodologies, this research follows the standard methodologies of the current fields of SPL and SWS practice. McGregor (2004) notes how agile development methodologies can be part of successful product line organizations. Further to this, Test Driven Development (TDD) is employed as well by incorporating a unit testing framework. Combining the best overlapping features of agile development and TDD, the facilities for prototyping and iteration among the prototypes ensures this research proceeds in a timely manner. Tools and frameworks common with SPL and also SWS are employed as well while utilizing these software development methodologies. The prototyping and testing framework will have to connect the feature model environment with the WSMO ontologies and the WSML descriptions of the services in the SOA. It must be able to support the reasoning requirements for verification of the product configurations developed in the feature model environment.

**ECLIPSE.** In the creation of any application or tools, as interim prototypes or final deliverables for this research, the Eclipse framework is utilized (http://www.eclipse.org). It is an open source open development platform that makes use of extensible frameworks, tools, and other supporting components. Eclipse-based tools commonly support SPL practices, see Spinczyk and Beuche (2004) for an example, in addition to being the basis for the tools listed above, specifically the Feature Plugin
and ATL. Foster, Uchitel, Magee, and Kramer (2005) demonstrate an Eclipse-based plug-in that can generate required orchestration (with BPEL) and choreography (with WS-CDL) specifications in the engineering of Web service compositions, leveraging many aspects of the Eclipse development platform. Lemmens, Wytzisk, de By, Granell, Gould, and van Oosterom (2007) utilize Eclipse in their prototype application, along with the OWL-S editor components from Protege, to assist in their ontology editing and service chaining work to improve on Web service discovery and composition. Utilization of the Eclipse platform ensures a strong supporting community to assist in any work in addition to making it likely that there is a ready audience of potential users to take advantage of any useful tools developed. Eclipse organizes its application framework around interoperable plug-ins. There is a wide variety of plug-ins designed for the Eclipse environment and they accomplish a diversity of tasks. This research project makes use of a number of existing Eclipse-based plug-ins including the ontology environment, the feature model environment, and the transformation functionality. WSMT, the Feature Plugin, ATL, and TCS are existing plug-ins used in this research and are described below. Any Java code development, including test case Web services and implementations of \texttt{ie.deri.wsmx.adapter}\texttt{.Adapter} necessary for lifting and lowering operations, is also conducted in the Eclipse environment.

**FEATURE PLUGIN.** The implementation of the feature model will make use of the open source Feature Modeling Plugin (http://gsd.uwaterloo.ca/projects/fmp-plugin) developed by the Generative Software Development Lab at the University of Waterloo. The Feature Plugin is based on the Eclipse Modeling Framework and implements cardinality-based feature modeling (Antkiewicz and Czarnecki 2004). Configurations created in the Feature Plugin can be used as input to code generating utilities or could even be treated by a system as a runtime configuration. From the view of the user of the prototype and testing environment, modeling will occur from this perspective. Figure 9 illustrates
the main user interface of the Feature Modeling Plugin perspective in Eclipse.

Figure 9. Feature Model in the Feature Model Plugin

The Feature Plugin also supports, although in a very limited implementation, an XML serialization of the feature model structure, and example of which is shown in Listing 1. This is made available through the XML Export command. However, in the current version of the Feature Plugin, there is no working implementation of a complimentary XML Import utility. Further, even the current version of XML Export does not export all elements of the feature model. For example, there are currently no serializations for the requires and excludes constraints, as well as any extensions to the basic feature model metamodel which a developer may implement. While it would be possible to extend this tool and create a customized XML Export tool, fortunately this is not required. The .fmp file itself of the Feature Plugin is Ecore compatible and can be read and navigated directly by ATL.
Unfortunately though, the metacircular nature of the implementation makes navigation of the feature model somewhat challenging and the solutions to these challenges are shown in the detailed design.

Listing 1. Feature Model XML Export

```xml
<feature min="1" max="1" name="VirtualWholesaleComposition" type="NONE" id="composition">
  <feature min="0" max="1" name="Producers" type="NONE" id="producers">
    <featureGroup min="1" max="1" id="group">
      <feature min="0" max="1" name="AtlanticTech" type="NONE" id="atlanticTech">
      </feature>
      <feature min="0" max="1" name="NorthernSystems" type="NONE" id="northernSystems">
      </feature>
      <feature min="0" max="1" name="InfoDesigns" type="NONE" id="infoDesigns">
      </feature>
      <feature min="0" max="1" name="SystemImports" type="NONE" id="systemImports">
      </feature>
    </featureGroup>
    <feature min="0" max="1" name="Shippers" type="NONE" id="shippers">
      <featureGroup min="1" max="1" id="group0">
        <feature min="0" max="1" name="CanadaPost" type="NONE" id="canadaPost">
        </feature>
        <feature min="0" max="1" name="Purolator" type="NONE" id="purolator">
        </feature>
        <feature min="0" max="1" name="Midland" type="NONE" id="midland">
        </feature>
        <feature min="0" max="1" name="FederalExpress" type="NONE" id="federalExpress">
        </feature>
      </featureGroup>
      <feature min="0" max="1" name="Insurers" type="NONE" id="insurers">
        <featureGroup min="1" max="1" id="group1">
          <feature min="0" max="1" name="SouthwestInsurance" type="NONE" id="southwestInsurance">
          </feature>
          <feature min="0" max="1" name="PatriotInsurance" type="NONE" id="patriotInsurance">
          </feature>
          <feature min="0" max="1" name="DirectCoverage" type="NONE" id="directCoverage">
          </feature>
        </featureGroup>
        <feature min="0" max="1" name="PaymentMediators" type="NONE" id="paymentMediators">
          <featureGroup min="1" max="1" id="group2">
            <feature min="0" max="1" name="Visa" type="NONE" id="visa">
            </feature>
            <feature min="0" max="1" name="Mastercard" type="NONE" id="mastercard">
            </feature>
          </featureGroup>
        </feature>
      </featureGroup>
    </feature>
  </feature>
</feature>
```
Although WSMO ontologies are typically written in WSML, it is possible to export the information to an XML format, WSML-in-XML, which is used by WSMO Studio. It is also possible to represent feature model in an XML format. This has been illustrated in the sections above. Being in XML, and with full understanding of the metamodels, the available tools of model transformation become accessible to this work. An initial possibility is to utilize, for the transformation, the Extensible Stylesheet Language Transformations (XSLT). XSLT is an XML-based language that allows the translation of XML files from one format or schema to another. However, the use of XSLT may not be ideal for this work. Moran and Mocan (2005) note that XSLT is rooted in syntax and has an inability to fully account for semantics. Jovanovic and Gašević (2005) identified XSLT transformations as being very sensitive to the format of the input document structure and therefore the transformations can be difficult to maintain as formats evolve. Finally, XSLT deals with extremely verbose XML syntax and this would appear to stray from the principles of MDD in terms of reducing the overall complexity to the developers.

As a better alternative, this research has adopted the ATLAS Transformation Language (ATL). Milanovic et al. (2007) demonstrate the use of ATL and identify its open source status, large user community, solid developer support, and wide variety of example applications to be valuable assets. Transformations in ATL are in the form of modules containing a header, an import section, operation and attribute helpers (OCL based), and transformation rules. ATL requires valid metamodels in order to conduct transformations and transformations can only occur in one direction. Metamodels can be created within ATL using the KM3 format. KM3, the Kernel Metametamodel, is a syntax similar to Java notation, for describing a metamodel that, once serialized into XML, is suitable for supporting model transformation in ATL. A good description of model transformations in ATL is found in Jouault.
and Kurtev (2006). Declarative (i.e. matched) rules are primarily used although the imperative features that ATL supports are utilized where appropriate. ATL exists as a plug-in to Eclipse and is therefore an ideal component to integrate with this prototype and testing framework.

In order to use ATL for this transformation, a metamodel for WSMO needed to be created in the KM3 format that ATL requires. The metamodel incorporated all relevant components of WSMO elements: Ontologies, Goals, and Web Services. Mediators were not implemented due to their incomplete specification and the lack of any need to use mediators in this work. Coding the metamodel in KM3 proved to be a fairly time consuming exercise to ensure that all elements of the fairly expressive WSMO metamodel were represented. Furthermore, certain custom elements needed to be added to the model so that the transformation could derive the textual concrete syntax WSML file. This was required to facilitate TCS which is described below. Although there is a WSML-in-XML format specification already (and it is used in WSMO Studio), it is not sufficient for the purposes of this work. It does not implement a complete interface description. Not only does it not implement orchestration (which is typical among existing WSMO implementations), but it also does not implement full choreography descriptions either. According to discussion on the WSMO Studio online discussion group (Dimitrov 2008), the WSML-in-XML format is not used very much.

**WSMT.** After transformation, there does need to be some manual intervention of the resulting WSML output. There are a number of tools which could be utilized for this from WSMO Studio to generic text editors to the Web Service Modeling Toolkit (WSMT). WSMT (http://sourceforge.net/projects/wsmt) is chosen due to its integration with the WSMX Execution Environment. WSMT is able to parse .wsml files and verify syntax. It is also able to conduct initial discovery and data mediation work with WSMT or even directly link to a running WSMX engine to upload or download .wsml files.
**WSMX EXECUTION ENVIRONMENT.** The WSMX execution environment (http://www.wsmx.org/) is a reference implementation of WSMO which facilitates the discovery, invocation, composition, and monitoring of Web services described in WSMO. It is an essential component to this work. It utilizes a number of different reasoners (IRIS, KAON2, etc.), implements various discovery engines (keyword, lightweight, and heavyweight), and allows for data and process mediation. Cimpian and Mocan (2006) describe process mediation in WSMX which is based on choreographies. It allows two public processes to be combined to achieve a certain functionality and utilizes the Communication Manager and Choreography Engine components to mediate between the two processes, coordinating the input and output of their transition rules. If different ontologies are used between these two processes, data mediation using the Data Mediator component can be invoked. Though designed to manage real world SWS implementations, when WSMX is called through an appropriate testing framework, the output from its various components provides valuable feedback to a developer conducting feature modeling tasks on SOA. Web services can only be discovered and invoked by the use of WSMO Goals so it is important that for any orchestrating Web service which is created by the transformation, there must be an associated Goal which is capable of discovering it. Although only choreography (choreography.wsmx) is implemented fully in WSMX, there is a stand-alone orchestration component (orchestration.wsmx) which is provided with the current version of WSMX for unit testing purposes. WSMX does require certain rules be followed which constrain the WSML Web service descriptions it can execute. In addition to specific non functional properties which must be included to facilitate Web service discovery and lifting/lowering during process mediation, there is a limited set of transition rule constructs which can be interpreted by WSMX. Furthermore, it is necessary to add code to WSMX itself in order to conduct lifting and lowering between the SOAP-based XML messages and actual WSML. The existing lifting and lower adapters in WSMX are all
specific to the examples and it is necessary to implement the ie.deri.wsmx.adapter.Adapter interface provided by WSMX.

**JUNIT TESTING FRAMEWORK.** The JUnit testing framework (http://www.junit.org/) is used from the Eclipse environment to take advantage of the existing testing classes provided with the WSMX source. These can be modified from the Eclipse Java perspective with little effort to then simulate the full functionality of the WSMX execution engine and provide the appropriate feedback to the developer.

**WEB SERVICE DEVELOPMENT.** This work requires a set of fully functional Web service test cases which can be used for discovery, invocation, and composition. These SOAP-based Web services were created in Java utilizing the jaxws-api and jaxb-api JARs. Web services were loaded and deployed through a Web service container, Apache Tomcat (http://tomcat.apache.org/).

**WSMO Studio.** Based on the Eclipse platform, WSMO Studio is an integrated modeling environment to work within the WSMO framework (Dimitrov, Simov, Momentchev, and Konstantinov 2007). Its wide ranging functionality includes core components to create and validate WSMO models, import and export from and to WSML, RDF, and OWL formats, as well as a fully featured WSMO editor, choreography editor, and SAWSDL editor. Like WSMT, it has integrated reasoners, including Pellet, MINS, IRIS, and KAON2. Some of the reasoners are geared only towards descriptive logic and therefore only a subset of the WSML languages, while others are capable of reasoning against more WSML variants. While WSMO Studio does support service discovery, this is a keyword-based QoS discovery engine only and is not sufficient for the purposes of this work. WSMO Studio is explored and referenced for comparative purposes only.

**DEVELOPMENT ENVIRONMENT.** The primary development environment used is a Dell Dimension 2400 workstation running Linux (VectorLinux 5.8, with KDE), though development and
testing is also conducted in parallel on a Windows notebook as well. WSMT v.1.4.4 is installed for manual intervention on .wsml files and the WSMX execution environment v.0.5 is employed. For feature modeling, Eclipse v3.2, along with EMF v.2.2.0 to support Feature Plugin v.0.6.6 is used although Feature Plugin v.0.7.0, which is still considered in development, is tested as well. Transformation is done using ATL v.2.0 and extraction to WSML is done using TCS v.0.8.0. There are issues with some of the utilities functioning well only within certain versions of Eclipse. This does make creation of a single comprehensive plugin solution difficult to achieve but this does not detract from the research value of the findings. Porting certain tools such as the Feature Plugin to newer builds of Eclipse is beyond the scope of this work.
CHAPTER 4
DETAILED DESIGN

The detailed design of this integrated project incorporates a number of enhancements to existing methodologies and technologies. From introducing a standardized methodology for representing variability of SOA within a feature model, to the heuristics and patterns required to migrate the information to the WSMO context for reasoning purposes, there are a number of specific design stages in the project to detail within this chapter. Figure 10 below outlines the overall steps in the process according to the detailed design.

Figure 10. Process Supported by the Detailed Design
4.1 REPRESENTATION OF VARIABILITY

There are numerous ways to represent variability within a feature model diagram. However, in order to support transformation from the feature model formalism to that of a Web service composition in WSMO, it is necessary to make certain assumptions about the feature model and then enforce specific methods to represent the variability within it. In order for feature models to represent Web service composition, we need to introduce of the conceptual ordering of services to the practice of feature modeling. Although the order of features on the same level within a feature model typically has no real meaning, in this case the developer must follow a certain style to ensure that the feature model is reflective of the general order of the potential flow of the business process. This order will of course be reflected in greater detail within the Web service composition description in WSMO, which is based on abstract state machines (ASM) as described in the previous chapter. The style incorporates the feature modeling methodology proposed by Montero et al. (2008) as an initial base. Their basic composition rules are illustrated in Figure 11. It is the application and extension of these basic rules which will govern the transformation design described in later sections.

Figure 11. Basic Feature Model Methodology

Figure 11 relates the structure and order of business processes, in our case represented by SWSs, within a feature model to their transition rules in the ASM which form essential elements of the Choreography and Orchestration descriptions in WSMO. In Figure 11, item 1 shows that service A will
fire and change the state of the ASM before service B can fire. In item 2, service A will fire, changing the state of the ASM and service B may or may not be required or able to fire. In item 3, service A will fire and change the state of the ASM and then both services B and C are able to fire in parallel. In item 4, service A fires, changing the state of the ASM and then either B or C, but not both, fire. In item 5, service A fires, changing the state of the ASM and then one or both of services B and C are able to fire.

However, this work implements some enhancements to the basic principles identified by Montero et al. (2008) for a number of reasons. The first alteration is to ensure the existing feature modeling environment is utilized to manage its own constraints, which the Feature Plugin does quite well, so therefore only the product configuration itself, which represents a potential concrete composition of SWSs, is actually required to be exported to the WSMO environment for reasoning. There is no significant added value in exporting the entire structure of the feature model to the WSMO environment at this stage in modeling when what is actually desired is the composition of SWSs to reason against. This results in a much smaller set of mapping rules. Furthermore, there is already the work of Wang et al. (2007) which facilitates ontology-based verification of the feature models themselves, as well as product configurations against the feature models. However, in our case, the feature modeling environment already serves to enforce the main constraints of the feature model against derived product configurations and also provides significant reasoning feedback for the developer in its own right. This is optimized for this environment and therefore the better focus is on the potential to add value beyond that, primarily to provide reasoning on the developer's proposed composition, in the form of a product configuration, against real-world ontologies, Web services, and goals (as well as mediators at some point). Therefore, the product configuration of the feature model contains no unresolved variability and represents a concrete composition of SWSs to be implemented within the WSMX execution environment.
The second alteration is to incorporate an additional level of abstraction by taking advantage of the two main types of elements available through the Feature Plugin – Features and Feature Groups. Presumably most, if not all, variability within a SOA can be represented by sets of grouped features within a hierarchy, leaving solitary features outside of these groups to represent higher levels of abstraction. Features which represent actual SWSs will therefore always be contained within feature groups representing the appropriate cardinality information. In addition to facilitating the higher level of abstraction to assist the developer, these assumptions also allow the transformation developed by this project to better navigate the feature model. In the example feature model in Figure 12, the parent features of the feature group represent categories of services. The child features of the feature group represent actual services which can comprise the composition. The constraints, requires and excludes, function as they would normally function within the Feature Plugin. These constraints only affect what
can be done during product configuration but, once resolved, there is no need to preserve that information beyond the valid product configuration to the WSMO environment.

Finally, this work makes use of the ability in the Feature Plugin to apply attributes to the various features. This extension to the basic feature model metamodel is very useful in that it can allow the easy inclusion, at design time, of information which may be more difficult to derive through other means. For the purposes of this research, the attributes included with the services represented as features are the real-world groundings of the input the actual Web service expects. These are actual URIs which tend to be very long Web addresses and are used to invoke a Web service. Output groundings are not necessary as these will actually be handled by the execution environment. For developers to take full advantage of the transformation developed by this work, it is important to annotate the feature model with these groundings.

The product configuration therefore contains a feature model tree with no unresolved variability. A possible product configuration of the above feature model is shown in Figure 13.

Figure 13. Example Product Configuration
The resulting composition would have an orchestration of Web Service B firing and changing the state of the ASM, followed by Web Service E doing so, and finally Web Service H. The higher level features, above the respective feature groups, provide the abstraction for the developer to group the features logically.

### 4.2 Mapping and Transformation

Using the ATLAS Transformation Language (ATL) in the Eclipse environment, this design maps the features in a feature model to associated molecular Web services which will become part of a composite Web service. Further to this, the resulting product configuration from the feature model is mapped to the structure of the actual composite Web service description in WSML. As ATL works on models and WSML descriptions are in text form, this integrated project utilizes TCS (Textual Concrete Syntax), also within the Eclipse environment, to develop a means by which to transform between the model representation of WSMO to the textual concrete syntax of WSML.

Unfortunately, orchestration is not currently implemented within the WSML language standard or the WSMX execution environment. Current tool support within the WSMO family cannot even parse full orchestration descriptions at this time. Presently we see in the literature, such as the work of Gone and Schade (2008) and Brambilla et al. (2007), a necessary but unfortunate misuse of choreography to achieve roughly the same effect as orchestration. This can be done fairly easily. However, this likely poses certain limitations on the current ability of WSMO tools to orchestrate composite services 'on the fly' and this will be discussed in greater detail in Chapter 6. There is an early version of an orchestration.wsmx component which, while not yet integrated into the WSMX environment, currently exists for unit testing. However, the orchestration description in WSML is still at the stage of a final draft and not at the stage of an accepted standard of WSMO, although it adopts the same ASM approach as WSMO choreographies and incorporates state signatures and transitions.
rules. While orchestration does best represent the relationships that a feature model is capable of incorporating; for the purposes of this work, only choreography is a product of the transformation. Orchestration possibilities will be discussed in Chapter 6 but, in order to ensure that the output of the transformation is executable in WSMX; orchestration descriptions will not be created and are outside of the immediate scope of this work.

We also need to create, as a separate artifact, a WSMO Goal which is capable of being used to discover and invoke the composite service in the WSMX environment. Currently, WSMX compositions must be engaged through use of a WSMO Goal which describes the service enough that it can be discovered, invoked, and has the ability to govern process mediation with it. For the purposes of testing potential compositions and extracting information from the reasoning functionalities in the WSMX environment, it is necessary to create and use this goal. It is a further verification on the transformation to test that the composite services it creates are, in fact, capable of being discovered by the various discovery engines that WSMX employs. However, the WSMO Goal that will call the composite service can be virtually identical to the composite service itself for these purposes. There are a few minor changes to the output Web service description that can be deferred to the manual intervention to accomplish this.

The heuristics of the transformation involve considering the order of the selected features in the product configuration to populate certain sections of the composite service's WSML description, specifically the capability and interface information. The capability is represented through preconditions and postconditions, as well as assumptions and effects not actually implemented in this work. There are not currently any WSMO implementations which reason against assumptions and effects. WSMX ignores them entirely and, considering they refer to facts outside the immediate information space of the Web service composition, they are outside the scope of this work. As in
Brambilla et al. (2007), the preconditions of the composite service would generally be (though not necessarily exclusively) all preconditions of the first service(s) called in the composition. The postconditions of the composite service would generally be (though not necessarily exclusively) all postconditions of the last service(s) called in the composition. Where necessity requires deviation from this heuristic, such as for certain side-effects of a composition, additional manual intervention could be required. This could be required to account for very complex pre- and post-conditions but this is not the case for the test cases in this work or typical in examples in the literature. The interface description of the composite service is comprised of choreography and orchestration descriptions. Both choreography and orchestration descriptions require state signatures (though they can be similar) which define the state of the ASM through a set of mode containers. These containers have different modes including static, in, out, shared, and controlled. Generally, within this design, containers with an in mode represent input to the composite web service (and therefore input to its molecular services) while those with an out mode represent output from the composite web service (and therefore output from its molecular services). The transformation will migrate the appropriate in and out mode containers to the composite Web services. Other mode containers, such as those with a controlled mode are used to help control flow within the ASM and can either be assumed for the purpose of the transformation or managed through manual intervention. Orchestration and choreography descriptions also require transitions rules. There is a generally accepted pattern, represented by the Ontologized Abstract State Machine (Herold 2008), which is known to the WSMX choreography engine, to manage some of the control from within the ASM that can be assumed by the transformation. However, for choreography, the details within each transition must be extracted from the transition rules of the molecular services' choreographies. It is important to note that, while the outside guards of the guarded transitions may be the same between the orchestration and choreography descriptions, that actual rules within the
transitions will be fundamentally different. Transition rules in the choreography manage the input and output of information to and from the composite web service and do not directly reference its molecular services, only their representative mode containers within the state signature. On the other hand, an orchestration can explicitly call the individual molecular services. As the output from the feature model is, at its simplest, an ordered listing of service invocations, an orchestration would be much easier to derive. A choreography is more difficult to arrive at and is therefore the source of some manual intervention following the transformation. Table 1 summarizes the mapping rules applied in the transformation.

Table 1. Summary of Basic Mapping Rules

<table>
<thead>
<tr>
<th>Feature Model Element</th>
<th>Description</th>
<th>WSML</th>
</tr>
</thead>
<tbody>
<tr>
<td>![Example Feature Model]</td>
<td>Root Project</td>
<td>Not Directly Mapped (Contains Feature Model, Metamodel, and Metametamodel elements of .fmp file)</td>
</tr>
<tr>
<td>![Example Composition]</td>
<td>Root of Feature Model</td>
<td>webService ExampleComposition nonFunctionalProperties dc#title hasValue ExampleComposition dc#description hasValue ExampleComposition endNonFunctionalProperties capability ExampleCompositionCapability interface ExampleCompositionInterface choreography ExampleCompositionChoreography stateSignature ExampleCompositionSig transitionRules ExampleCompositionTrules</td>
</tr>
<tr>
<td>![Category of Services 1]</td>
<td>Selected Service (single group)</td>
<td>// Pre and Post Conditions Inside WSMO Capability precondition definedBy.// TODO: Preconditions of ServiceB postcondition definedBy.// TODO: Postcondition of ServiceB // Out Mode Container in State Signature out ServiceBResponse //with optional grounding ... // ServiceB Web Service Transition forall {?controlstate} with( ?controlstate[oasm#value hasValue oasm#InitialState] memberOf oasm#ControlState) do</td>
</tr>
</tbody>
</table>
### Feature Model Element | Description | WSML
---|---|---
for all with() do // TODO: Service A Related Specific Transition Rules end forall delete(?controlstate[ oasm#value hasValue oasm#InitialState] memberOf oasm#ControlState) add(?controlstate[ oasm#value hasValue oasm#State1] memberOf oasm#ControlState) end forall

---

**Selected Services** *(Two Consecutive Groups)*

// Pre and Post Conditions Inside WSMO Capability

precondition definedBy. // TODO: Preconditions of Service A

postcondition definedBy. // TODO: Postcondition of Service C

// Out Mode Container in State Signature

out ServiceAResponse //with optional grounding …

out ServiceCResponse //with optional grounding …

// Transition Rules

forall {?controlstate} with(?controlstate[ oasm#value hasValue oasm#InitialState] memberOf oasm#ControlState) do
for all with() do

// TODO: Service A Related Transitions
end forall

delete(?controlstate[ oasm#value hasValue oasm#InitialState] memberOf oasm#ControlState)

add(?controlstate[ oasm#value hasValue oasm#State1] memberOf oasm#ControlState)
end forall

forall {?controlstate} with(?controlstate[ oasm#value hasValue virtual#State1] memberOf oasm#ControlState) do
for all with() do

// TODO: Service C Related Transitions
end forall

delete(?controlstate[ oasm#value hasValue virtual#State1] memberOf oasm#ControlState)

add(?controlstate[ oasm#value hasValue oasm#State2] memberOf oasm#ControlState)
end forall

---

The transformation must navigate the input model, in this case the Ecore-compatible Feature Model file itself (the Feature Plugin's .fmp file simply renamed to .ecore), in order to obtain the
elements that will be created in the output model. A further step is the necessity of converting the output model to WSML text, which is done using TCS. The navigation of the input model must consider the meta-circular nature of the Feature Model metamodel where the base element Feature is used to represent all types of features in the tree, regardless of position or function in the model. Whether a feature is a root feature, mandatory, optional, solitary, or grouped, it will be stored in the metamodel simply as a feature. In some cases, the type of feature can only be derived from attributes of the parent feature group. Furthermore, the metamodel itself (and the metametamodel) is embedded as a Feature Model within the .fmp file. Any given feature model is simply a product configuration of the metamodel, which is itself structured as a feature model implementing the metametamodel. All features are children of the single root element Project. Figure 14 illustrates this view through the Ecore Model Editor available with the ATL package in Eclipse.

Figure 14. Model, Metamodel, and Metametamodel in the .fmp File

This makes navigation of the feature model somewhat challenging to ensure that the transformation does not process any features which contribute no relevant information relating to the potential Web service composition the developer is intending the reason against. Other than the root Project element which contains no information, the only other element type in the metamodel is the
Feature Group. Considering the conceptual mappings defined earlier, it is clearly important to be able to know, while processing a feature, if it is part of a feature group. Given the metamodel of the Feature Plugin, the only way to find this out is to navigate to the parent and query its attributes. Fortunately for these challenging aspects of the transformation, the ATL language does allow for Helper functions which can be used to easily query various elements, including information regarding parent and child features, as well as assisting the navigation of the overall structure. The ability to determine whether a feature is grouped or not can simply be obtained by calling the helper function in Listing 2 from any transformation rule acting on features.

Listing 2. Feature Group Helper Function

```
-- Helper function to check if the parent is a Feature Group
helper context FMIFeature def: isNotParentGroup(): Boolean =
    self.refImmediateComposite().oclType().toString() <> 'FMIFeatureGroup';
```

Similarly, the ability to determine whether or not a feature being navigated is part of the metamodel or metametamodel trees is determined through the following helper in Listing 3.

Listing 3. Metamodel Helper Function

```
-- Helper function to ensure not working with Metamodel features
helper context FMIFeature def: isNotMeta(): Boolean =
    if (self.refImmediateComposite().oclType().toString() <> 'FMIProject') then
        if self.refImmediateComposite().oclType().toString() <> 'FMIFeatureGroup' then
            self.refImmediateComposite().isNotMeta()
        else
            (self.refImmediateComposite()).refImmediateComposite().isNotMeta()
        endif
    else
        if (self.name <> 'MetaModel' and self.name <> 'MetaMetaModel') then
            true
        else
            false
        endif
    endif;
```

Other required helpers used to ensure accurate navigation and query of the input model include those which differentiate between top-level features (i.e. their parent is the Project element), which
differentiate between product configurations and the base feature model, and whether the features in the product configuration are selected, not selected, or undecided. All these helper functions are essential in navigating the Feature Plugin input and these can be found within the source listing of the .atl file in Appendix A. These helpers are called during navigation of the input model through the various rules, lazy rules, and lazy unique rules of the ATL transformation. They can be used to create a subset of the input features to pass along to a rule so that the rule only has to deal with the appropriate features, such as in the ATL code in Listing 4.

Listing 4. Calling Helper Functions

```plaintext
GroundFacts <- (FM!Feature.allInstances() -> select(f|f.isNotMeta() and f.isConfiguration() and f.isSelected() and f.isNotUndecided()) -> collect(e| thisModule.FM2(e)))
```

This use of the helpers eases the navigation of the metacircular structure of the input from the Feature Plugin so that the rules can focus on the ordering of actual services within the model. The ATL code in Listing 4 calls the lazy rule FM2, in Listing 5 which focuses on extracting the in mode containers for the state signature. Lazy rules are called indirectly and return the transformation back to the main matched rule which called them upon completion. They are necessary in this transformation to conduct different tasks on the same set of features without disrupting the overall order of the transformation matched rules. A simplified listing of this rule is shown in Listing 5. The full listing is contained within the .atl file in Appendix A.

Listing 5. Lazy Rule

```plaintext
lazy rule FM2 {
  from
    b : FM!Feature (b.isParentGroup() and b.isNotUndecided() and b.isConfiguration() and b.isNotTreeRoot() and b.isNotMeta() and b.isSelected())
  to
    p : WSMO!GroundFacts(
      iri <- attr,
      Grounding <- attr2),
    attr : WSMO!Basic_IRI ( 
```

63
Creation of output features (ATL does not actually directly navigate the output model) is predicated against ensuring there is a metamodel for the output. Currently, there is no publicly available metamodel file for WSMO in the appropriate KM3 format required by ATL for the purposes of this transformation. Even if there had been, it would likely have required significant modification in order to make it compatible with the TCS aspects of the transformation. There are specific patterns required by TCS which must be followed for viable translation into the WSML concrete syntax. The KM3 file describing the WSMO metamodel is presented in Appendix B. While creating a TCS for a given domain specific language such as WSMO is a time-consuming exercise, it provides little research value in its own right. Nonetheless it is essential to the development of this integrated project and is provided in Appendix C. The transformation utilizes most of the information present in the product configuration to derive an outline for a composite Web service description in WSML. However, manual intervention is then necessary before this composition is able to be discovered and invoked within the WSMX execution environment so valuable reasoning-based information can be provided to the developer regarding the potential composition.

4.3 MANUAL INTERVENTION

The initiation transformation does not result in a completely executable composite Web service description. Complete automation of such a transformation is likely not achievable in the short term. This is partly due to the currently available tools in the WSMO environment, the lack of specification for some aspects of WSMO, as well as the limited information which can be stored in the feature model formalism. Feature models being less expressive than WSML Web service descriptions generally, there
will likely be need in the foreseeable future for significant manual intervention to create compositions which can be discovered and invoked within an execution environment like WSMX.

The first aspect of manual intervention involves those related to Non-functional Properties (NFP). Typically used for metadata annotation purposes, NFPs are also essential for discovery in WSMX. They are also necessary in directing the Invoker component of WSMX to the appropriate lifting and lowering adapter to be used for transforming messages between SOAP-based XML of the Web Service and WSML required by WSMX. The use of NFPs can indicate to the WSMX environment which discovery engines should or should not be used in the discovery of services against a given goal. Should certain keyword, lightweight (postconditions only), or heavyweight (preconditions and postconditions) discovery engines prove more or less suitable in a given situation, it is up to the developer to provide this guidance through discovery related NFPs to ensure appropriate discovery success. The framework developed by Steinmetz, Kerrigan, Lausen, Tanler, and Sirbu (2008) shows how NFP’s in WSMO Web service descriptions can be used to direct the execution environment to the use of the appropriate discovery engine which should be used to best discover that service. NFPs in WSMO Goal descriptions can also be used to direct the execution environment to the preferred discovery engines. Furthermore, within the JUnit test environment associated with WSMX that is used by this design, the default initial (i.e. first pass) discovery process is based on keyword compatibility and therefore it is important to ensure similar NFP information with appropriate keywords between the output Web service and the related WSMO Goal which will be used to discover and invoke the composition—a necessity with the WSMX execution environment. Generally, the transformation inserts appropriate NFP information from the names of root features used in the feature model which should enable appropriate keyword discovery. However, should WSMX not discover Web services in the initial first pass, there may be manual interventions required to fine-tune this. WSMX also requires
the "http://owner" NFP to be present in any executable Web service description in order to guide the Invoker component to the appropriate lifting and lowering adapter. All Web services utilized in this work contain an "http://owner" NFP with a value of "http://io.acad.athabascau.ca/~jeffr. This directs the Invoker component to utilize the com.comp699.adapter.Comp699Adapter specifically designed to lower WSML requests to the expected XML in the SOAP-based Web services and lift the XML response from these Web services to the WSML required by WSMX.

There is likely also a requirement, in most cases, to manually check and revise the control flow within the transition rules. This is necessary because the transitions as described within the molecular services, and incorporated into the composite services, may need some modification to be compatible with the OASM which is used to govern the transition from state to state within the transition rules and also to reflect any additional complexity in the transition rules of the molecular services. As requests and responses pass between the various molecular services, which are not otherwise aware of each other, it falls to the composite service to direct WSMX to resolve any inconsistencies between expected and actual messages. While WSMX does support data and process mediation, the composite Web service needs to utilize the appropriate shared variables in the transition rules which match appropriate concepts in the ontologies imported by the services. Examples of this are shown in the demonstration in Chapter 5.

Other minor requirements for manual intervention include checking for consistency among namespaces. The various molecular services which make up a composition may reference many different namespaces and import different domain and process ontologies. It is important to ensure the compositions created also reference the relevant namespaces where appropriate. While the transformation can simply incorporate all namespace references from the molecular services into the composite service, this will likely require checking before any assumptions can be made regarding the
various references made to these namespaces. Another source of manual intervention could be the adding of appropriate comments to the WSML, though the transformation does add some useful comments. However, this is not a key factor to the use of the output from the transformation at this time and comments are only added manually for the sake of readability. To make the work of manual intervention easier, the transformation does insert specific \texttt{TODO} comments where manual intervention is required to ensure appropriate execution within WSMX. Finally, but essential, is the extraction of a WSMO Goal which is used to discover and invoke the composite Web service. Provided the appropriate manual intervention has been done with the NFPs of the composite Web service as detailed above, it can be as simple a matter as changing the description from \texttt{webService} to \texttt{goal}.

With this goal there also is required a simple instance ontology, trailing the goal description, which includes any specific user input required to pass to the input services. Table 2 provides a summary of all the typical components of a complete, executable WSML Web service description and where this process obtains the information.

Table 2. Summary of Origin of WSML Web Service Elements

<table>
<thead>
<tr>
<th>WSMO Component</th>
<th>Source</th>
</tr>
</thead>
<tbody>
<tr>
<td>wsmlVariant</td>
<td>Transformation (utilizes constant in ATL)</td>
</tr>
<tr>
<td>namespace</td>
<td>Transformation (utilizes constants in ATL)</td>
</tr>
<tr>
<td>webService</td>
<td>Transformation</td>
</tr>
<tr>
<td>nfp</td>
<td>Transformation (utilizes constants in ATL) and Manual Intervention</td>
</tr>
<tr>
<td>importsOntology</td>
<td>Transformation and Manual Intervention</td>
</tr>
<tr>
<td>capability</td>
<td>Transformation</td>
</tr>
<tr>
<td>sharedVariables</td>
<td>Manual Intervention</td>
</tr>
<tr>
<td>precondition</td>
<td>Manual Intervention</td>
</tr>
<tr>
<td>postcondition</td>
<td>Manual Intervention</td>
</tr>
<tr>
<td>interface</td>
<td>Transformation</td>
</tr>
<tr>
<td>choreography</td>
<td>Transformation</td>
</tr>
<tr>
<td>WSMO Component</td>
<td>Source</td>
</tr>
<tr>
<td>--------------------</td>
<td>----------------------------------</td>
</tr>
<tr>
<td>stateSignature</td>
<td>Transformation</td>
</tr>
<tr>
<td>in mode containers</td>
<td>Transformation</td>
</tr>
<tr>
<td>out mode containers</td>
<td>Transformation</td>
</tr>
<tr>
<td>controlled mode containers</td>
<td>Transformation</td>
</tr>
<tr>
<td>transitionRules</td>
<td>Transformation and Manual Intervention</td>
</tr>
<tr>
<td>OASM Ontology</td>
<td>Transformation and Manual Intervention</td>
</tr>
</tbody>
</table>

### 4.4 CONFIGURATION OF WSMX

There are some steps required to prepare WSMX for use as the execution environment for any set of Web services. All Web services, ontologies, and goals used must be able to be found by the execution environment. WSMX can be directed to these WSMO elements by listing their locations in the `config.properties` file. The resource manager component of WSMX utilizes this file in the loading of WSMO resources. Listing 6 shows how this information is entered into the `config.properties` file for resources for the TestWSML project directory. All semantically described resources used in the Demonstration in Chapter 5 need to be available to WSMX through this listing.

**Listing 6. Direction in config.properties File to Available WSMO Resources**

```properties
wsmx.resourcemanager.goals = \n"$(resources)/resourcemanager/TestWSML/Goals" ;
wsmx.resourcemanager.goals.offline = \n"$(resources)/resourcemanager/TestWSML/Goals" ;
wsmx.resourcemanager.ontologies = \n"$(resources)/resourcemanager/TestWSML/Ontologies" ;
wsmx.resourcemanager.webservices = \n"$(resources)/resourcemanager/TestWSML/SWSs" ;
```

As mentioned in the previous section, there also needs to be appropriate lifting and lowering adapters for any Web services used by WSMX. These adapters should implement the ie.deri.wsmx.adapter.Adapter interface provided by WSMX and requires separate functions to conduct the specific lifting (from the SOAP-based XML to WSML) and lowering (from WSML to SOAP-based
XML) operations necessary for process mediation to take place during execution. For the purposes of this work, a custom adapter which implements the ie.deri.wsmx.adapter.Adapter interface expected by WSMX was written in Java. For any additions of adapters beyond the ones provided by the WSMX examples, it is necessary to rebuild WSMX from source to ensure this Adapter is available to the Invoker component. As mentioned in the manual intervention section, all Web services utilized in this work contain an "http://owner" NFP with a value of "http://io.acad.athabascau.ca/~jeff" which directs the Invoker component to use the com.comp699.adapter.Comp699Adapter. The code for this adapter is available in Appendix D.

4.5 REASONING

Reasoning is achieved by engaging the WSMX execution environment with the proposed composite service and its associated goal. This is actually done through the JUnit testing framework included with the WSMX source, WSMXExecution.java, and then modifying the provided Java test class, org.deri.wsmx.unittest.main.AchieveGoalExecutionSemanticsTest.java which implements junit.framework.TestCase. This modification is shown below in Listing 7. This simulates the discovery, data mediation, and process mediation steps of the WSMX execution environment without the full WSMX environment running. When run as a JUnit test, it loads the listed goal and the associated instance ontology and runs WSMX through the discovery, data mediation, and process mediation components required to fulfill that goal. All input is output to the console for the developer to view and the overall result of the JUnit test assertion is returned. This conducts the process exactly as the full WSMX execution environment would run, but within the development environment and without the overhead of the full execution environment running.
Listing 7. Example *AchieveGoalExecutionSemanticsTest.java*

```java
package org.deri.wsmx.unittest.main;
import java.util.List;
...
import org.deri.wsmx.unittest.util.WSMXExecution;
import org.wsmo.execution.common.exception.ComponentException;

/**
  * Main off-server test. Reflects the behaviour of AchieveGoal Execution Semantics
  * run in WSMX Server mode. Facilitates testing and debugging process of WSMX
  * components and use-cases.
  */
public class AchieveGoalExecutionSemanticsTest extends TestCase {
    protected static Logger logger = Logger.getLogger(AchieveGoalExecutionSemanticsTest.class);
    public static WSMXExecution run = new WSMXExecution();

    public void testExampleFM() throws UnsupportedOperationException, ComponentException {
        String goalIRIStr = "http://www.example.org#ExampleFM";
        String goalOntoIRIStr = "http://www.example.org#ExampleFMOntology";
        List<String> result = run.runDiscoveryDataMediationAndChoreography(goalIRIStr, goalOntoIRIStr);
        assertTrue(result.size() != 0);
    }
}
```

The type of information that is returned to the developer during execution of the composition is diverse. The first information is whether the WSML description of the WSMO Goal (and therefore likely also the Web service description it is derived from) and its related instance ontology can be located by the environment and parsed. The WSMX environment loads all WSMO elements it can obtain as listed in the file *config.properties*. If a WSMO element is found and loaded by WSMX, this will be noted in the initial output. For example, provided *config.properties* has a link to the resources/resourcemanager/testing/SWSs directory, valid Web services within that directory will be loaded into WSMX as show in Listing 8.

Listing 8. Examples as Loaded into WSMX

```
INFO InMemoryRM : resources/resourcemanager/testing/SWSs
INFO InMemoryRM : Stored webservice http://www.example.org#WebServiceA from WebServiceA.wsml
```

If it is not possible to parse the WSML file, this will be detected at the startup of the WSMX
execution environment as in Listing 9 (with a simple spelling mistake on WSML syntax).

Listing 9. Failure to Parse

```
WARN Helper : Failed to parse
C:\Software\Eclipse_New\eclipse\Workspace\WSMX\resources\resourcemanager\testing\SWS\PayVisa.wsml.
```

Assuming we have valid WSML descriptions of all relevant elements, the next step by WSMX is to load the WSMO Goal and use it to discover the relevant composite service. The Goal looks very similar to the Web service description with similar (if not identical) non functional properties to aid in the initial keyword-based discovery and then appropriate postconditions in the capability through which the lightweight reasoner can reason against services which may fulfill this postcondition. The Goal also requires a state signature providing the appropriate in and out mode containers to be used in process mediation with this Goal and any Web services it would employ. An example of such a Goal is provided in the demonstration in Chapter 5. Whether any discovery is possible or not is important to the developer and gives an indication as to whether further manual intervention is necessary to improve the Web service description itself (or also the problem could be with the goal description, but not likely considering the derivation of the goal from the Web service). If the Web service cannot be discovered by a WSMO Goal that is almost an exact match in terms of description than the developer needs to look closely at the composite Web service and, possibly, NFPs related to discovery. If discovery does not occur, the feedback as shown in Listing 10 is returned.

Listing 10. WSMX Output with Lack of Discovery

```
INFO WSMXExecution : STARTED AT 12:28:20
INFO WSMXExecution : Retrieved Goal: http://www.example.org#ExampleFMGoal
ERROR WSMXExecution : Goal Ontology has not been found: http://www.example.org#ExampleFMGoal
INFO AchieveGoalChor : Discovering suitable webservices for: http://www.example.org#ExampleFMGoal
INFO KeywordDiscovery : Keyword discovery - operating on 32 services.
INFO AchieveGoalChor : Discovery did not result in any candidate services.
INFO WSMXExecution : TIME TAKEN: 1761 ms.
INFO WSMXExecution : FINISHED AT 12:28:22
```
If potential Web services can be found, we have entirely different output. Listing 11 demonstrates how the keyword-based discovery finds suitable services and then there is the employment of the Lightweight discovery (based on postconditions) which finds an exact match. Although preconditions and assumptions are ignored by WSMX as shown in the listing, the lightweight discovery engine successfully uses the postcondition of the services found by the keyword-based discovery engine to discover an exact match in terms of output.

Listing 11. Successful Discovery

```
INFO  WSMXExecution            : STARTED AT 09:04:31
INFO  WSMXExecution            : Retrieved Goal: http://www.example.org#ExampleGoal2
INFO  WSMXExecution            : Retrieved Goal Ontology: http://www.example.org/ExampleGoal2Ontology
INFO  AchieveGoalChor          : Discovering suitable webservices for: http://www.example.org#ExampleGoal2
INFO  KeywordDiscovery         : Keyword discovery - operating on 2 services.
WARN  LWRuleDiscovery          : Preconditions and assumptions will be ignored supported
WARN  LWRuleDiscovery          : Preconditions and assumptions will be ignored supported
WARN  LWRuleDiscovery          : Preconditions and assumptions will be ignored supported
INFO  LWRuleDiscovery          : Comparing goal with Web service: http://www.example.org#ExampleService1
INFO  LWRuleDiscovery          : Comparing goal with Web service: http://www.example.org#ExampleService2
INFO  LWRuleDiscovery          : Exact match
```

If the keyword-based discovery is able to operate on any services then we have a good idea that the appropriate NFPs have been included into the Web service description. If there is an exact match possible then we know there are valid postconditions in the Web service description which match the Goal. With an exact match that should be expected at this point, the next reasoning process is that of data mediation. The ontologies must either be compatible or there needs to be data mapping files for the various ontologies available to the WSMX execution environment. Unfortunately the output from the WSMX execution environment is verbose but there are some key elements of the output which can report valuable information back to the developer. Listing 12 provides sample output related to data mediation.
Equally verbose is the output from the process mediation. However, this is just as important to the developer evaluating the suitability of the proposed composite web service. Listing 13 illustrates some initial sample output from the WSMX execution environment showing step by step information regarding the execution of the choreography.

Listing 13. Abbreviated Choreography Output

A successful choreography ends with mediated instances of the concept types originally requested by the goal returned to the goal. Listing 14 shows typical output upon successful process mediation.

Listing 14. Successful Output Instance Data
Even when successful process mediation does not occur, there is extensive, although relatively verbose, feedback provided by the WSMX execution environment for the developer to evaluate what possibly went wrong with the composition and at what stage in execution. Inability to mediate between data, incompatible choreographies, and unavailable services can all be communicated to the developer. This output is obtained in the demonstration detailed in Chapter 5 and subsequently analyzed for the purposes of evaluating the reasoning potential in Chapter 6.
CHAPTER 5
DEMONSTRATION

The demonstration of this integrated project creates and utilizes a number of simple WSMO services which are loaded into the WSMX environment so they can be discovered and composed based on the design information of the feature model transformed into the WSMO formalism. The demonstration consists of these WSMO services, a feature model incorporating these services, an example of a product configuration composing these services, the transformation, and some modifications to the JUnit testing framework in the WSMX source which allow testing the discovery and process mediation to engage the reasoner and provide useful information to the developer. Figure 15 provides a schematic of the steps conducted in this demonstration and is followed by the running case used to provide a hypothetical context for the demonstration.

Figure 15. Steps of Demonstration

5.1 RUNNING CASE

The running case is comprised of a hypothetical virtual wholesale business-to-business
environment. There are a number of producers, shippers, insurers, and payment mediators involved as potential actors in the business process. Producers make available certain goods at specific prices, shippers will ship those goods at specific rates, insurers will insure the value of both produced goods and shipment costs, while finally payment mediators will return an approval or disapproval on the final amount of the transaction (including product costs, shipping costs, and insurance costs). The domain is kept very simple to ensure that the focus is on representing the variability of the business process within a feature model and the resulting configuration into an executable Web service composition to be reasoned upon. Producers will return the price based on the quantity of product ordered while shippers will return a shipping rate based on the quantity of product ordered as well. More complex factors such as weight, shipping method, expediency, etc. are beyond the scope of this simple demonstration. Insurers return an insurance cost based on fixed rates while payment mediators are simply determined based on the total cost (i.e. costs above a certain amount will not be approved). All producers, shippers, insurers, and payment mediators are in the form of Web services described in the following section.

5.2 WEB SERVICES

Web services, one for each of four producers, four shippers, three insurers, and two payment mediators, have been created. All Web services are SOAP-based services written in Java, utilizing jaxws-api and jaxb-api JARs. All source code is available in Appendix E. The simple Web services are composed of an interface which they implement, the Service Endpoint Interface (SEI), and the actual Service Enterprise Bean (SIB). These artifacts are compiled by the Java compiler and then the wsgen utility is run to create the JAX-B artifacts required to generate the Web service's WSDL. A WEB-INF directory is created to place custom configuration files web.xml and sun-jaxws.xml which are deployment descriptors required to allow the Web service to be deployed through a Web service.
container in this case Apache Tomcat. At this point, the folder for the Web service (in this case the AtlanticTech producer Web service) has the files and subdirectories shown in Listing 15.

Listing 15. Contents of Web Service Java Package Folder

```
\wsmo\atlantictech\AtlanticTech.class
\wsmo\atlantictech\AtlanticTech.java
\wsmo\atlantictech\AtlanticTechImpl.class
\wsmo\atlantictech\AtlanticTechImpl.java
\wsmo\atlantictech\jaxws
\wsmo\atlantictech\WEB-INF
\wsmo\atlantictech\jaxws\ReturnPrice.class
\wsmo\atlantictech\jaxws\ReturnPriceResponse.class
\wsmo\atlantictech\WEB-INF\classes
\wsmo\atlantictech\WEB-INF\sun-jaxws.xml
\wsmo\atlantictech\WEB-INF\web.xml
\wsmo\atlantictech\WEB-INF\classes\wsmo
\wsmo\atlantictech\WEB-INF\classes\wsmo\atlantictech
\wsmo\atlantictech\WEB-INF\classes\wsmo\atlantictech\AtlanticTech.class
\wsmo\atlantictech\WEB-INF\classes\wsmo\atlantictech\AtlanticTechImpl.class
\wsmo\atlantictech\WEB-INF\classes\wsmo\atlantictech\jaxws
\wsmo\atlantictech\WEB-INF\classes\wsmo\atlantictech\jaxws\ReturnPrice.class
\wsmo\atlantictech\WEB-INF\classes\wsmo\atlantictech\jaxws\ReturnPriceResponse.class
```

All the above files are archived into a Web archive file (.war) using the jar utility. The resulting .war file is then placed into the webapps directory of Apache Tomcat. When the Web services are deployed through Tomcat, their associated WSDL files can be viewed through a Web browser at a link such as, for the Atlantic Tech producer, http://localhost:8080/atlantictech/atlantictech?wsdl. This represents the address of the Web service and will be required in adding the grounding. Listing 16 illustrates the WSDL file for the Atlantic Tech producer Web service.

Listing 16. WSDL File for Atlantic Tech Producer Web Service

```
  targetNamespace="http://atlantictech.wsmo/" name="AtlanticTechImplService">
  <types>
    <xsd:import namespace="http://atlantictech.wsmo/"
```
The WSDL file describes the Web service and how to interact with it from a syntactic perspective. However, there is no real meaning which can be derived from this information without applying semantics through ontology concepts. To do this, it is necessary to create semantic Web service descriptions for these services and relate them to concepts in existing domain ontologies. However, first, it is necessary to look at the actual messages that the Web services expect to receive and return as output in order to create adapters for WSMX. Using the tool SOAPUI (http://www.soapui.org), it is possible to look at the actual XML messages that govern the request and response of these Web services. Listing 17 shows the request XML message for the Atlantic Tech producer Web service while Listing 18 shows the response XML message from the Atlantic Tech
producer Web service. These messages are used to guide development of the lifting and lowering adapters described in Chapter 5.

Listing 17. Input Message to Atlantic Tech Producer Web Service

```xml
  <soapenv:Header/>
  <soapenv:Body>
    <atl:returnPrice>
      <arg0/>
    </atl:returnPrice>
  </soapenv:Body>
</soapenv:Envelope>
```

Listing 18. Output Message from Atlantic Tech Producer Web Service

```xml
<S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/">
  <S:Body>
    <ns2:returnPriceResponse xmlns:ns2="http://atlantictech.wsmo/">
      <return/>
    </ns2:returnPriceResponse>
  </S:Body>
</S:Envelope>
```

With Web services deployed and lifting/ lowering adapters created, it is then required to semantically describe the Web services. Given the simple Web services, it is not an overly complex task to create these descriptions; however it is a time consuming manual operation. Non functional properties aid in discovery operations and associate the Web services with specific adapters. Input and output groundings are very important and relate directly to the actual location of the Web service and its WSDL file. The pre- and post-conditions semantically describe the input and output of the Web service in relation to concepts in the imported domain ontology. Finally, the transition rules are created which describe how the inputs and outputs of the Web service are managed through process mediation. Listing 19 provides an example for the Atlantic Tech producer Web service. All other Web services descriptions, including the existing domain ontologies they reference can be found in Appendix F.
Listing 19. Semantic Web Service Description for Atlantic Tech Producer Web Service

```xml
wsmlVariant _"http://www.wsmo.org/wsml/wsml-syntax/wsml-flight"
namespace {_"http://io.acad.athabascau.ca/~jeffr#",
                                      dc _"http://purl.org/dc/elements/1.1#"}
webService AtlanticTech
  nfp
dc#title hasValue "AtlanticTech"
dc#description hasValue "AtlanticTech"
  _"http://owner" hasValue _"http://io.acad.athabascau.ca/~jeffr"
endnfp
importsOntology MainOntology
capability AtlanticTechCapability
sharedVariables {?response, ?request}
precondition getQuantity
  definedBy
    {?request memberOf order.
postcondition returnPrice
  definedBy
    {?response [product hasValue "Atlantic"] memberOf ProducerResponse.
interface AtlanticTechInterface
choreography AtlanticTechChor
stateSignature AtlanticTechSig
importsOntology ProducerOntology
  in ProducerRequest withGrounding
    /returnPrice/returnPrice)"
  out ProducerResponse withGrounding
    /returnPrice/returnPriceResponse)"
transitionRules AtlanticTechTrans
  forall {?request} with {?request memberOf ProducerRequest)
  do
    add {?request memberOf ProducerResponse)
endForall

5.3 FEATURE MODELS AND PRODUCT CONFIGURATIONS

With Web services deployed and their semantic descriptions available to WSMX, the molecular
services are organized into a feature model. These services as features, although simple, represent real
grounded services requiring input and producing output which can reasonably contribute to a
composition to solve a greater problem.

80
Figure 16. Demonstration Feature Model

![Feature Model Diagram]

The feature model for the demonstration SOA is modeled as in Figure 16. The grounding information is truncated to fit the graphic into the space available. The variability is contained within the grouped features which represent actual Web services and can be used in potential compositions. The higher level parent features allow the developer a measure of abstraction to intuitively name the categories appropriately. As feature modeling is primarily a manual activity requiring active intervention from the developer, it is important to preserve relevant category information regarding the variability and generally ensure the developer has sufficient information required to configure the desired product. The feature model also makes use of an extension to the basic feature model metamodel by utilizing the Attribute property which allows a specific attribute to be associated with features. In this case, the in mode container groundings are stored in this property. These grounding
attributes are precise addressing information which can invoke the service and tend to be fairly long
with potential for error if this is not incorporated once at design time and instead left for repeated
manual intervention. These groundings, for example http://localhost:8080/infodesigns/infodesigns
?wsdl#wsdl.interfaceMessageReference(InfoDesigns/returnPrice/returnPrice), will ensure the resulting
product configuration contains the required grounding information to invoke any molecular services
selected. On the other hand, out groundings are not necessary to preserve in this manner as the WSMX
execution environment will handle all output in the process mediation.

For the purposes of this demonstration, the derived product configuration selects simply one
service from each category group (e.g. 1 producer from Producers group) to incorporate into a potential
composition. In fact, it is of course possible to select numerous services, running in parallel with their
siblings within their feature group, to create a more complex business process. An example of this
could be a requirement to ship products through two different shippers simultaneously. While this
would not change the basic principles applied here, it would potentially result in greater manual
intervention to manage the control flow of the ASM and is therefore beyond the scope of this integrated
project. Furthermore, most examples of WSMO compositions reviewed in Chapters 2 and 3
demonstrate a similarly linear approach to composition and it is likely that further improvements to
existing tool support are required before more complex compositions are common. It is realistic to
assume that most developers would be at first focused on compositions in such a manner. Nonetheless,
the scalability of this approach and others will be discussed in Chapter 6.

The example product configuration shown in Figure 17 attempts to create a valid composition
where the AtlanticTech producer Web service is fired and, having changed the state of the ASM, the
Midland shipper Web service is fired, then subsequently the DirectCoverage insurer Web service is
engaged, followed by the Mastercard payment mediator Web service. It will be this information which
is transformed into the outline of a composite WSMO Web service description, including its capability and interface description. Figure 18 shows the resulting composition logic.

Figure 17. Demonstration Product Configuration

Figure 18. Composition Logic
5.4 TRANSFORMATION

The output from the transformation creates a WSMO model. This can be viewed by ATL through the Ecore Model Editor. A portion of this can be seen in Figure 19.

Figure 19. Output Ecore WSMO Model

However, this is not actually WSML which can be loaded into the WSMX Execution Environment. To obtain the WSML, we utilize TCS and using the appropriate context menu we can Extract WSMO Model to WSMO File (.wsml) which creates the actual textual WSML file. This transformation creates the WSML Web service description for the composite Web service which incorporates the application logic of Figure 18. The actual output from the transformation (prior to any manual intervention) is provided in Listing 20 below.
Listing 20. Output WSML Listing of Composite Service

```
wsmlVariant _ "http://www.wsmo.org/wsml/wsml-syntax/wsml-flight"

webService VirtualWholesaleComposition

// TODO: Need to add http://owner NFP for adapter selection in WSMX
nonFunctionalProperties
   dc #title hasValue VirtualWholesaleComposition
   dc #description hasValue VirtualWholesaleComposition
endNonFunctionalProperties

importsOntology _ "http://io.acad.athabascau.ca/~jeffr#virtual"
// TODO: Import Any Domain Ontologies Used

capability VirtualWholesaleCompositionCapability
   sharedVariables {} // TODO: Any Shared Variables of Capability
   precondition definedBy. // TODO: Preconditions of First Service
   postcondition definedBy. // TODO: Postcondition of Final Service

interface VirtualWholesaleCompositionInterface

   choreography VirtualWholesaleCompositionChoreography
   stateSignature VirtualWholesaleCompositionSig

importsOntology // TODO: State Signature Requires Process Ontology

in MastercardRequest withGrounding _ "http://localhost:8080/mastercard/mastercard?wsdl#wsdl.interfaceMessageReference(Mastercard/returnApproval/returnApproval)"
out AtlanticTechResponse //with optional grounding ...
out MidlandResponse //with optional grounding ...
out DirectCoverageResponse //with optional grounding ...
out MastercardResponse //with optional grounding ...
// control flow containers
controlled oasm#ControlState

transitionRules VirtualWholesaleCompositionTrules
// AtlanticTech Web Service
forall (?controlstate) with(
   ?controlstate[oasm#value hasValue oasm#InitialState memberOf oasm#ControlState)
```

85
do
  forall with()
  do
    // TODO: AtlanticTech Related Transition Rules
  endForall
delete(?controlstate[oasm#value hasValue oasm#InitialState] memberOf oasm#ControlState)
add(?controlstate[oasm#value hasValue oasm#State1] memberOf oasm#ControlState)
endForall

// Midland Web Service
forall (?controlstate) with(
  ?controlstate[oasm#value hasValue virtual#State1] memberOf oasm#ControlState)
do
  forall with()
  do
    // TODO: Midland Related Transition Rules
  endForall
delete(?controlstate[oasm#value hasValue virtual#State1] memberOf oasm#ControlState)
add(?controlstate[oasm#value hasValue oasm#State2] memberOf oasm#ControlState)
endForall

// DirectCoverage Web Service
forall (?controlstate) with(
  ?controlstate[oasm#value hasValue virtual#State2] memberOf oasm#ControlState)
do
  forall with()
  do
    // TODO: DirectCoverage Related Transition Rules
  endForall
delete(?controlstate[oasm#value hasValue virtual#State2] memberOf oasm#ControlState)
add(?controlstate[oasm#value hasValue oasm#State3] memberOf oasm#ControlState)
endForall

// Mastercard Web Service
forall (?controlstate) with(
  ?controlstate[oasm#value hasValue virtual#State3] memberOf oasm#ControlState)
do
  forall with()
  do
    // TODO: Mastercard Related Transition Rules
  endForall
delete(?controlstate[oasm#value hasValue virtual#State3] memberOf oasm#ControlState)
add(?controlstate[oasm#value hasValue oasm#State4] memberOf oasm#ControlState)
endForall

ontology _"http://io.acad.athabascau.ca/~jeffr#virtual"
// TODO: oasm#ControlState concepts up to oasm#EndState

Of course, this is only a partially complete Web Service description in WSML and there needs
to be some manual intervention, as described in the detailed design of Chapter 4, before this is
executable within the WSMX environment.

5.5 MANUAL INTERVENTION

All instances of required manual intervention are annotated by ‘TODO’ comments in the output WSML file. These can be seen in Listing 20 above. In actual development practice, it would also be important to double-check some of the outputs of the transformation. However, for the purposes of this demonstration there are only a few items to check for consistency. Upon checking that the namespaces and discovery-related NFPs are consistent, which in this simple demonstration they are, we need to add the _"http://owner" NFP and assign a _"http://io.acad.athabascau.ca/~jeffr" value to it. This is not incorporated directly into the transformation to reflect the fact that custom adapters in WSMX are likely required for lifting and lowering. Should either the NFP or the actual adapter associated with the NFP not be present, there would be error messages returned from WSMX. This manual intervention ensures that the appropriate lifting and lowering adapter is utilized during process mediation. Other NFPs can be added at this stage but they are not necessary for the purposes of making the Web service executable in WSMX.

Some manual intervention is also necessary to ensure the control flow of the ASM in the choreography description is accurate and representative of the requirements of the services in the composition. An inspection of the molecular services is still necessary to ensure that the individual choreographies are satisfied in the composition choreography. In the case of this demonstration, it was necessary to make slight manual edits to the transition rules to account for the specific attributes in the process ontologies of the individual services. Any special data or process mediation must still be done by the composite service. For example, the payment mediator requires the aggregate of all three prices returned by the selected producers, shippers, and insurers respectively. Details of choreography related to business process within the molecular services is not represented in the feature model itself and
therefore must be retrieved from the semantic description of that actual service and often modified slightly for composition. To illustrate this example, in order to ensure that all three of these values are provided to the payment mediator, we see the following addition in Listing 21.

Listing 21. Manual Intervention for Payment Mediator Process

```owl
forall (?request1, ?input1, ?request2, ?input2, ?request3, ?input3)
with (?request1[value hasValue ?input1] memberOf ProducerResponse and
  ?request2[value hasValue ?input2] memberOf ShipperResponse and
  ?request3[value hasValue ?input3] memberOf InsurerResponse)
do
  add( #[price1 hasValue ?input1, price2 hasValue ?input2, price3 hasValue ?input3] memberOf PaymentRequest)
endForall
```

In order to utilize the OASM, which is known by the WSMX choreography implementation, a state ontology to follow the Web service description must be created with custom states between oasm#InitialState and oasm#FinalState. In this case, a state ontology must be created which reflects the number of states which exist in the composition to control the flow. This internal ontology typically trails the Web Service description in the same WSML file and defines the additional states that the ASM can have. Provided the oasm namespace is included in the WSML, we have the following addition in Listing 22 to accommodate the 2 additional states between the initial state and the final state. This small ontology provides to WSMX the concepts which guide the execution of the overall business process from producers (oasm#InitialState) to shippers (virtual#State1) to insurers (virtual#State2) to payment mediators (oasm#FinalState). WSMX is aware that oasm#InitialState represents the start of a choreography and when it encounters oasm#FinalState it ceases execution.

Listing 22. Additional States of the ASM

```owl
ontology "http://www.example.org/virtual"
  concept virtual#State1 subConceptOf oasm#ControlState
  concept virtual#State2 subConceptOf oasm#ControlState
```
There are also a few minor interventions required within the capability to ensure that postconditions (although part of the transformation, WSMX does not account for preconditions) of the last service, and any shared variables necessary to describe the condition, are included in the description. Any domain and process ontologies required by the molecular services should also be imported by the composite service so it is aware of all concepts used. At this point, we have a Web service description which, once loaded into the WSMX environment, can be discovered and invoked to compose a set of services to achieve a larger goal. This complete Web service description is shown below in Listing 23.

Listing 23. Web Service Description Following Manual Intervention

```xml
wsmlVariant "http://www.wsmo.org/wsml/wsml-syntax/wsml-flight"
namespace { "http://io.acad.athabascau.ca/~jeffr#",
    dc "http://purl.org/dc/elements/1.1#",
    oasm "http://www.wsmo.org/ontologies/choreography/oasm#"
}

webService VirtualWholesale

nfp
dc#title hasValue "VirtualWholesale"
dc#description hasValue "VirtualWholesale"
"http://owner" hasValue "http://io.acad.athabascau.ca/~jeffr"

endnfp

importsOntology {MainOntology, virtual}
capability VirtualWholesaleCapability

sharedVariables {?response, ?request}

precondition getQuantity
    definedBy
    {?request memberOf order}.

postcondition returnApproval
    definedBy
    {?response memberOf PaymentResponse}.

interface VirtualWholesaleInterface

choreography VirtualWholesaleChor
```
stateSignature  VirtualWholesaleSig
importsOntology  (ProcessOntology)
in  ProducerRequest  withGrounding
in  ShipperRequest  withGrounding
in  InsurerRequest  withGrounding
in  PaymentRequest  withGrounding
out  ProducerResponse
out  ShipperResponse
out  InsurerResponse
out  PaymentResponse
// control flow containers
controlled oasm#ControlState

transitionRules  VirtualWholesaleTrans

// AtlanticTech Web Service
// Midland Web Service
forall (?controlstate) with (  
  ?controlstate[oasm#value hasValue oasm#InitialState] memberOf oasm#ControlState )
do
  forall (?request, ?input) with (?request[quantity hasValue ?input] memberOf ProducerRequest)
do
    add(_# memberOf ProducerResponse)
    add(_#[quantity hasValue ?input] memberOf ShipperRequest)
  endForall
  delete(?controlstate[oasm#value hasValue oasm#InitialState])
  add(?controlstate[oasm#value hasValue State1])
endForall

// DirectCoverage Web Service
forall (?controlstate) with (  
  ?controlstate[oasm#value hasValue State1] memberOf oasm#ControlState )
do
  forall (?request1,?input1,?request2,?input2) with (?request1[value hasValue ?input1]  
    memberOf ProducerResponse and
  ?request2[value hasValue ?input2]  
    memberOf ShipperResponse)
do
    add(_#[price1 hasValue ?input1, price2 hasValue ?input2] memberOf InsurerRequest)
  endForall
  delete(?controlstate[oasm#value hasValue State1])
  add(?controlstate[oasm#value hasValue State2])
endForall

// Mastercard Web Service
forall (?controlstate) with ( ?controlstate[oasm#value hasValue State2] memberOf oasm#ControlState)
do
  forall (?request1,input1, ?request2, ?input2, ?request3, ?input3)
  with (?request1[value hasValue ?input1] memberOf ProducerResponse and
        ?request2[value hasValue ?input2] memberOf ShipperResponse and
        ?request3[value hasValue ?input3] memberOf InsurerResponse)
do
    add(_[price1 hasValue ?input1, price2 hasValue ?input2, price3 hasValue ?input3] memberOf PaymentRequest)
  endForall
delete( ?controlstate[oasm#value hasValue State2])
add( ?controlstate[oasm#value hasValue oasm#EndState])
endForall

ontology _http://io.acad.athabascau.ca/~jeffr#virtual
  concept State1 subConceptOf oasm#ControlState
  concept State2 subConceptOf oasm#ControlState
  concept State3 subConceptOf oasm#ControlState

Finally, the Goal used to discover and invoke the composite service must be created. The Goal should be very similar to the Web service description, though less information is required. Generally, the output of the transformation and initial manual intervention is sufficient, with minor revisions, to discover and invoke the composite Web service. By removing some information and adding an instance ontology with some input data trailing the Goal description, we have a Goal capable of discovering the newly created Web service, which is shown in Listing 24.

Listing 24. Goal to Discover and Invoke Demonstration Web Service

wsmlVariant _http://www.wsmo.org/wsml/wsml-syntax/wsml-flight
namespace { _http://io.acad.athabascau.ca/~jeffr#,
  dc _http://purl.org/dc/elements/1.1#"
}
goal VirtualWholesaleGoal
nfp
dc#title hasValue "VirtualWholesale"
dc#description hasValue "VirtualWholesale"
 _http://owner" hasValue _http://io.acad.athabascau.ca/~jeffr"
endnfp
importsOntology MainOntology
**capability** VirtualWholesaleGoalCapability

**sharedVariables** (?response, ?request)

**precondition** getQuantity
   
   **definedBy**
   
   ?request memberOf order.

**postcondition** returnApproval
   
   **definedBy**
   
   ?response memberOf PaymentResponse.

**interface** VirtualWholesaleGoalInterface

**choreography** VirtualWholesaleGoalChor

**stateSignature** VirtualWholesaleGoalSig

**importsOntology** {ProcessOntology}

  in ProducerRequest
  in ShipperRequest
  in InsurerRequest
  in PaymentRequest
  out ProducerResponse
  out ShipperResponse
  out InsurerResponse
  out PaymentResponse

**transitionRules** VirtualWholesaleGoalTrans

forall (?request) with (?request memberOf ProducerRequest)
   
   do
   
   add(?request memberOf PaymentResponse)
   
   endForall

**ontology** „http://io.acad.athabascau.ca/~jeffr#getVirtualWholesale”

**instance** myCompositionProduct memberOf ProducerRequest

  quantity hasValue 1

### 5.6 REASONING OUTPUT

The demonstration is completed by loading the files into the execution environment, including the reference to the Goal in the test case, and running the test case as a JUnit test. WSMX provides output information to the console. In this demonstration, we can see initially that all molecular services are able to be loaded into the WSMX execution environment and also that the composite service, VirtualWholesaleComposition, is loaded into WSMX. This means that there are no WSML parse errors
in the output of the transformation — including any errors related to namespace and Compact URI deficiencies. Listing 25 clearly shows that these service descriptions were loaded into WSMX. This is important information to the developer. However, if WSMX was unable to parse the composite Web service, there would be specific feedback detailing the errors.

Listing 25. Web Services are Available to WSMX

INFO InMemoryRM : resources/resourcemanager/TestWSML/SWSs
INFO InMemoryRM : Stored webservice http://io.acad.athabascau.ca/~jeffr#DirectCoverage from InsDirectCoverage.wsml
INFO InMemoryRM : Stored webservice http://io.acad.athabascau.ca/~jeffr#PatriotInsurance from InsPatriotInsurance.wsml
INFO InMemoryRM : Stored webservice http://io.acad.athabascau.ca/~jeffr#SouthwestInsurance from InsSouthwestInsurance.wsml
INFO InMemoryRM : Stored webservice http://io.acad.athabascau.ca/~jeffr#Mastercard from PayMastercard.wsml
INFO InMemoryRM : Stored webservice http://io.acad.athabascau.ca/~jeffr#Visa from PayVisa.wsml
INFO InMemoryRM : Stored webservice http://io.acad.athabascau.ca/~jeffr#AtlanticTech from ProdAtlanticTech.wsml
INFO InMemoryRM : Stored webservice http://io.acad.athabascau.ca/~jeffr#InfoDesigns from ProdInfoDesigns.wsml
INFO InMemoryRM : Stored webservice http://io.acad.athabascau.ca/~jeffr#NorthernSystems from ProdNorthernSystems.wsml
INFO InMemoryRM : Stored webservice http://io.acad.athabascau.ca/~jeffr#SystemImports from ProdSystemImports.wsml
INFO InMemoryRM : Stored webservice http://io.acad.athabascau.ca/~jeffr#CanadaPost from ShipCanadaPost.wsml
INFO InMemoryRM : Stored webservice http://io.acad.athabascau.ca/~jeffr#FederalExpress from ShipFederalExpress.wsml
INFO InMemoryRM : Stored webservice http://io.acad.athabascau.ca/~jeffr#Midland from ShipMidland.wsml
INFO InMemoryRM : Stored webservice http://io.acad.athabascau.ca/~jeffr#Purolator from ShipPurolator.wsml
INFO InMemoryRM : Stored webservice http://io.acad.athabascau.ca/~jeffr#VirtualWholesale from TestComposition.wsml
INFO InMemoryRM : Stored ontology http://io.acad.athabascau.ca/~jeffr#Virtual from TestComposition.wsml

With all services, and especially the composite service created by the transformation (and subject to the manual intervention described earlier), loaded successfully into WSMX, the execution engine proceeds to the discovery portion of the testing. This provides important feedback to the developer. If the composite Web service is not capable of being discovered, even by a Goal which is described in a very similar manner, there needs to be some additional manual intervention or there may be other things requiring attention. The Web service description may require more detailed description, additional NFPs, or other information included. In any case, this is valuable information to the developer. The detailed output of WSMX allows a view into the discovery process and the ability of the newly created composition to participate in this. Listing 26 below illustrates how the keyword-based
discovery engine was able to include the composite VirtualWholesaleGoal in the initial listing of services to be applied to the lightweight discovery engine. When the appropriate service is finally selected, its choreography is loaded and registered. This lets the developer know that the choreography is valid to the extent that WSMX can load it and understand its state signature and transition rules so that data mediation can begin.

Listing 26. Successful Discovery and Loading of Choreography

```
INFO WSMXExecution : Retrieved Goal: http://ioacadathabascauca/~jeffr#VirtualWholesaleGoal
INFO WSMXExecution : Retrieved Goal Ontology: http://ioacadathabascauca/~jeffr#VirtualWholesaleGoal
INFO AchieveGoalChor : Discovering suitable webservices for: http://ioacadathabascauca/~jeffr#VirtualWholesaleGoal
INFO KeywordDiscovery : Keyword discovery - operating on 42 services.
INFO LightweightDiscovery : Testing Web service http://ioacadathabascauca/~jeffr#VirtualWholesale
INFO AchieveGoalChor : --- Performing Data Mediation; direction: GOAL_TO_WEBSERVICE
INFO AchieveGoalChor : """"Mediated instances """
INFO AchieveGoalChor : http://ioacadathabascauca/~jeffr#myCompositionProduct memberOf
INFO Radex : Registering requester choreography
INFO Radex : Registering provider choreography
INFO Radex : "id:http://ioacadathabascauca/~jeffr#VirtualWholesaleChor
INFO Radex : signature id:http://ioacadathabascauca/~jeffr#VirtualWholesaleSig
INFO Radex : rules id:http://ioacadathabascauca/~jeffr#VirtualWholesaleTrans
INFO LogicalState : Initializing state http://www.wsmx.org/choreography/state_id-0at3062331014347694380#onto
```

As mentioned in the detailed design, there needs to be groundings to real world services for complete process mediation to take place. As can be seen in the abbreviated Listing 27 below, the mode containers in the composite Web services are grounded to real-world (http://localhost:8080) services. As discussed earlier in Chapter 4, groundings to out mode containers are not necessary.

Listing 27. Groundings

```
INFO Radex : R-TO-P Groundings:
INFO Radex : http://ioacadathabascauca/~jeffr#myCompositionProduct has
INFO Radex : http://www.wsmo.org/reasoner/anonymous_1249566818571-1779884208 has
```

With or without actual grounded services though, WSMX will attempt process mediation. Looking at Listing 28 below, in the first line we see "STEP 1" which correlates to the Initial State
expected by WSMX in the OASM. WSMX will attempt to execute all transition rules when their guards allow them to be reached based on the input instance data provided by the Goal. The information provided through this to the developer is important and allows a view into the process mediation. At this point, the developer can see whether process mediation is at all possible and, if not, determine where errors occurred in the process.

Listing 28. Process Mediation

```
INFO Machine : STEP 1
INFO Machine : Executing "forall {?controlstate} with
(?controlstate[_"http://www.wsmo.org/ontologies/choreography/oasm#value" hasValue
"http://www.wsmo.org/ontologies/choreography/oasm#InitialState"] memberOf
"http://www.wsmo.org/ontologies/choreography/oasm#ControlState") do
  add(?controlstate [_"http://www.wsmo.org/ontologies/choreography/oasm#value" hasValue
"http://io.acad.athabascau.ca/~jeffr#State1"])
forall (?input,?request) with (?request[_"http://io.acad.athabascau.ca/~jeffr#quantity" hasValue ?input] memberOf
"http://io.acad.athabascau.ca/~jeffr#ProducerRequest") do
  add(_# memberOf "http://io.acad.athabascau.ca/~jeffr#ProducerResponse")
  add(_# [_"http://io.acad.athabascau.ca/~jeffr#quantity" hasValue ?input] memberOf
"http://io.acad.athabascau.ca/~jeffr#ShipperRequest")
endForall
delete(?controlstate [_"http://www.wsmo.org/ontologies/choreography/oasm#value" hasValue
"http://www.wsmo.org/ontologies/choreography/oasm#InitialState"])
```

Process mediation, which provides fairly verbose output to the console, executes the various states of the OASM in the transition rules. The output is verbose because all guarded transitions are evaluated at each stage to determine which rules will be executed. In the output, when actual molecular Web services are invoked, we can see the lowering and lifting processes as in Listing 29 below.

Listing 29. Invoking of Molecular Web Services

```
INFO AchieveGoalChor : ------------ Invoker: Sending WSML to service ------------
Service grounding: >>>
<<<
myCompositionProduct memberOf ProducerRequest
quantity hasValue 1
INFO Invoker : service: http://io.acad.athabascau.ca/~jeffr#VirtualWholesale
```
INFO Invoker : data size: 1
INFO Invoker : grounding:
porttype: AtlanticTech
operation: returnPrice
endpoint: http://localhost:8080/atlantictech/atlantictech

INFO Invoker:
----------Outgoing SOAP message----------
<?xml version="1.0" encoding="UTF-8"?>
   <SOAP-ENV:Header/>
   <SOAP-ENV:Body>
      <atl:returnPrice xmlns:atl="http://atlantictech.wsmo/">
         <arg0>1</arg0>
      </atl:returnPrice>
   </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
----------End of SOAP message----------

INFO Invoker:
----------Incoming SOAP message----------
<?xml version="1.0" encoding="UTF-8"?>
<S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/">
   <S:Body>
      <ns2:returnPriceResponse xmlns:ns2="http://atlantictech.wsmo/">
         <return>545.32</return>
      </ns2:returnPriceResponse>
   </S:Body>
</S:Envelope>
----------End of SOAP message----------

INFO AchieveGoalChor : ---------- Invoker: Received WSML from service -----------
templInst1249566819118 memberOf ProducerResponse
   value hasValue 545.32

In this demonstration, we can see in Listing 29 above that a quantity of 1 item (1 was the quantity in the original instance ontology of the goal) was passed to the AtlanticTech Producer Web Service through lowering the WSML instance to SOAP-based XML and the price of $545.32 was returned, through lifting the SOAP-based XML to WSML, to the process. The return is an instance of concept ProducerResponse with a value of $545.32 which will serve as input to other Web services further down in the business process. This shows successful invocation of the producer molecular service which was selected in the original feature model. Similar output from
the WSMX Invoker component is returned for the other services to demonstrate successful execution of the choreography.

With data mediation and process mediation successfully completed, there is final output response to the requestor (i.e. the Goal) in the form of an ontology. This concludes the operation of the WSMX execution environment and this is shown below in Listing 30. The instance of concept PaymentResponse is the answer from the payment mediator, in this case "true" indicated that payment is able to be processed.

Listing 30. Response Ontology

```
INFO AchieveGoalChor : ------------ Final response to requestor -----------
   namespace { wsml _"http://www.wsmo.org/wsml/wsml-syntax#" }
   ontology _"http://wsmx.org//responseOntology1783308446288430806"
   instance _"http://wsmx.org//responseOntology1783308446288430806#tempInst1249566819733_-731343865"
   memberOf _"http://io.acad.athabascau.ca/~jeffr#PaymentResponse"
   _"http://io.acad.athabascau.ca/~jeffr#value" hasValue "true"
INFO AchieveGoalChor: --- Performing Data Mediation; direction: WEBSERVICE_TO_GOAL
INFO AchieveGoalChor: ---------- Mediated instances ----------
INFO AchieveGoalChor: http://wsmx.org//responseOntology1783308446288430806#tempInst1249566819733_-731343865
memberOf http://io.acad.athabascau.ca/~jeffr#PaymentResponse
INFO WSMXExecution : TIME TAKEN: 3362 ms.
INFO WSMXExecution : FINISHED AT 10:53:39
```

The entire process completes successfully in 3362 ms. Although the positive output for a successfully execution is valuable information for the developer, this demonstration will also show some key negative information to inform the developer when successful execution with the selected services is not possible. With some simple examples, we can modify some key variables of the environment to illustrate this valuable information to the developer.

An initial example of unsuccessful execution would be one or more of the selected services not being available. This can be simulated by simply shutting down the Tomcat Web service container temporarily and re-running the execution. In this case, Semantic Web service descriptions are still available to WSMX but the real-world grounding is not valid. In Listing 31 below, we can see that the
connection to the actual AtlanticTech Web service is refused. The choreography attempts to still continue but at the end of the run, data mediation (Web Service to Goal) does not occur and no results are returned.

Listing 31. Connection to Actual Web Service Refused

```
INFO AchieveGoalChor: ----------- Invoker: Sending WSML to service -----------
Service grounding: >>>
myCompositionProduct memberOf ProducerRequest
quantity hasValue 1
INFO Invoker : service : http://io.acad.athabascau.ca/~jeffr#VirtualWholesale
INFO Invoker : data size : 1
INFO Invoker : grounding :
ERROR Invoker : Connection refused: connect
```

In terms of exercising some of the reasoning capabilities of unsuccessful executions, we can make some simple modifications to the domain or process ontologies which would result in an incompatible choreography. To do this, we simply change the PaymentRequest concept as a subConcept of the approval concept to a subConcept of the price concept, which would make payment mediation not compatible with which is expected in the transition rules. Again, at the end of the run, data mediation (Web Service to Goal) does not occur and no results are returned. In addition to returning an org.wsml.reasoner.api.inconsistency.InconsistencyException to the console indicating that the reasoner has detected an inconsistency in the underlying ontologies that the Web service refers to, WSMX indicates that there is no matching input and output between the Goal and the Web services that it has discovered and composed. Listing 32 provides some of the relevant information for this problem.

In terms of exercising some of the reasoning capabilities of unsuccessful executions, we can make some simple modifications to the domain or process ontologies which would result in an incompatible choreography. To do this, we simply change the PaymentRequest concept as a subConcept of the approval concept to a subConcept of the price concept, which would make payment mediation not compatible with which is expected in the transition rules. Again, at the end of the run, data mediation (Web Service to Goal) does not occur and no results are returned. In addition to returning an org.wsml.reasoner.api.inconsistency.InconsistencyException to the console indicating that the reasoner has detected an inconsistency in the underlying ontologies that the Web service refers to, WSMX indicates that there is no matching input and output between the Goal and the Web services that it has discovered and composed. Listing 32 provides some of the relevant information for this problem.
Listing 32. Inconsistent Ontologies Underlying the Composition

```java
org.wsml.reasoner.api.inconsistency.InconsistencyException: Consistency Violation detected! (first: AttributeTypeViolation due to instance: "http://io.acad.athabascau.ca/~jeffr#tempInst1249569414652" expected type: "decimal" found value: "true" at attribute: "http://io.acad.athabascau.ca/~jeffr#value")
```

There are a great many combinations which can be done to illustrate the reasoning feedback returned from the WSMX execution environment. The value of this information will be discussed in Chapter 6.
CHAPTER 6
RESULTS AND ANALYSIS

6.1 RESULTS

This work has demonstrated how it is possible to represent services in a SOA as a feature model and then transform the resulting product configuration into the Web Service Modeling Ontology for reasoning purposes. This reasoning information is intended to give the developer some insight into how the composition would function with real-world molecular services in an actual execution environment. While the feature modeling environment employed in this work, the Feature Plugin, ensures that product configurations are consistent with the feature model, it can provide no support in determining whether the product configuration is consistent with the real-world services and their choreographies. Actual SOAP-based Web services were first described semantically in WSML and then incorporated into a feature model to model the SOA as a SPL. With a feature model composed of molecular services, a resulting product configuration represents a proposed composite Web service and is the input to the transformation. The transformation demonstrates that most of the information inherent in the feature model formalism can be transformed into WSMO to generate usable artifacts and provide additional information to the developer through the reasoning engines that WSMX employs. With a simple set of rules applied to feature modeling practices, the transformation was able to extract the overall business process and other relevant information into a skeleton WSMO representation. With some manual intervention to complete the Web service description and ensure it meets the requirements of the execution environment, the composite Web service was loaded into WSMX and invoked through a corresponding goal and sample instance data. The demonstration completed successfully, indicating to the developer that it was in fact possible to compose the real-world services
selected from the feature model into an executable composition. The demonstration also tested alternative instances where the real-world services could not be composed into an executable composition. In these cases, as expected, composition could not be achieved and appropriate feedback was issued to the developer. In addition to the demonstration specific to Chapter 5, the solution was applied to other product configurations from the running case, all of which were successful (where the input was not modified to ensure unsuccessful completion) and returned valuable reasoning-based information to the developer.

The solution developed provides semantic reasoning-based information to the design and configuration of service-oriented software product lines. The reasoning output of WSMX, while verbose, returns significant reasoning-based information to the developer. This includes the results of keyword-based and lightweight discovery, data mediation, and process mediation—I all of which were illustrated in the demonstration. None of this information is available in the feature modeling environment alone. The lightweight discovery engine, once keyword-based discovery has been employed as an initial screen, reasons whether the stated postconditions of the goal are compatible with the postconditions of any Web service in the repository. Postconditions are required to be compatible in terms of concept. It is important to note that they do not have to be identical concepts, only equivalent, and the reasoning engine queries the ontologies to determine this. It also ensures that the concepts in the instance ontology used by the Goal to provide executable input data are consistent. Process mediation also conducts reasoning to ensure that all concepts of the in and out mode containers are of compatible concept type and can be satisfied by the choreography. Process and domain ontologies, where imported by Web service and Goal descriptions, are queried during this process. In addition to information provided by the reasoners employed, the demonstration also returns some run-time information which is returned to the developer regarding the validity of the transitions as it moves from
state to state. Finally, where there is real instance data as input, there is the returned output (as an ontology) to the user to confirm the successful execution of the composition and the accuracy of any returned value, in addition to its concept type.

Although it was expected that the less-expressive feature modeling notation would not fulfill all elements of an executable Semantic Web service description, the specific differences between expected and actual reasoning is still worth further consideration. The WSMX execution environment provides good feedback on the validity of the output from the transformation (i.e. can the WSML be parsed?) as well as the ability to be discovered with the various discovery engines employed by WSMX. Although only keyword-based and lightweight (postconditions only) discovery engines were used in the demonstration, similar feedback may be provided with the other discovery engines although the evaluation of these other discovery methods were beyond the scope of this work. Information related to data mediation was also returned during execution although, with all services using the same data and process ontologies, it was expected that data mediation would be fairly straightforward. Should data mediation have been more difficult, it would have been necessary to create ontology mappings files in a tool such as WSMT prior to this work and then evaluate how reasoning occurs against these data mapping files though WSMT does provide data mapping unit testing in its own environment. In terms of process mediation, the choreography description is reasoned against to ensure that the choreography is valid and WSMX provides relevant information when it is not. Groundings in the state signature are checked prior to execution and sufficient output to the console is returned during execution to assist in determining any problems with lifting and lowering of data. However, there is no opportunity to reason against any orchestration description, despite this potentially important representation of the business process. As current tool support for WSMO does not even parse orchestration descriptions, and this transformation does not output such descriptions, there is no way to
formally validate the orchestration beyond that done in the choreography process mediation. For the purposes of this running case, this is sufficient. In larger more complex cases, however, separate reasoning would likely be required for both choreography and orchestration. For this solution to at some point be applicable to the discovery and invocation of services unknown to the immediate WSMX environment or with much larger repositories, such reasoning would likely be essential. The current version of WSMX does contain a stand-alone component, *orchestration.wsmx*, which can be used for unit-testing of orchestration descriptions. This component, however, is not yet integrated into WSMX process mediation and, considering the WSML specification for orchestration is not yet finalized, is not demonstrated in this work. Without the ability to integrate and reconcile both choreography and orchestration descriptions in the process mediation, including the separate stand-alone test for orchestration description in the demonstration would not have added additional value in terms of reasoning-based information to the developer.

Although it was not possible to exercise aspects of WSMO orchestration descriptions in this work, it was possible to utilize the choreography description to achieve the same effect as orchestration. This was achieved through the grounding of the *in* mode containers directly to the WSDL of the actual SOAP-based Web services. This has the effect of directly invoking those services in the choreography. However, this is not actually the intended use for choreography, which is primarily to present the external means by which to communicate with the composite service. Due to the details required for choreography though, additional manual intervention was required to ensure that the order and types of input and output are reconciled from the original results of the transformation. With full implementation of orchestration at some point, it would likely be possible to represent the orchestration-based invocation of molecular Web services from the feature model and then a simpler external choreography could be added to the composite Web service description.
Overall, the solution was successfully applied to the running case as well as the minor deviations applied to the input to exercise other aspects of reasoning. This serves as a proof of concept that services organized in a feature model can be exported into a semantic reasoning environment for reasoning-based input at the design stage. However, simple successful execution is a very coarse grade by which to evaluate any solution and therefore the following section will look at the solution from a number of perspectives common to the evaluation of integrated software solutions.

6.2 EVALUATION

This section will evaluate the solution from a number of perspectives, specifically productivity, ease of use, scalability, and performance. These factors illustrate the viability of the overall solution and provide insight into the ways in which this solution can be implemented to contribute to both test and production environments.

PRODUCTIVITY. Applying MDD to this running case allows for a significant amount of the necessary Web service description to be created automatically from the product configuration to WSML. This automatic generation of WSML code can reduce error and development time for creating compositions and their descriptions. Creating semantic descriptions for Web services is a time-consuming task and with composite Web services there is even greater potential for error. Although WSML is relatively close to natural language, compared to OWL-S for example, it is nevertheless a potentially complex activity to semantically describe any Web service in terms of its capability and interface. Any automation which can be provided to the developer is useful. Combined with well-placed TODO comments in the resulting WSML, with this transformation the developer can more easily create executable compositions. The semantically described Web service WSML file serves as an actual composite service to the WSMX execution environment. Manual intervention was required in
this work and realistically it would likely not be possible to eliminate entirely manual intervention, especially as the choreography description must be used to achieve orchestration as well. Table 3 below outlines the average percentage of automatically generated portions of WSML description achieved in this work. Percentages were calculated considering the number of words of functional WSMO elements, and specific non functional elements required for WSMX execution, between the original output of the transformation and the subsequent executable version following manual intervention. All comments and other annotation, although important to the developer and valuable in their own right, were removed for this comparison.

Table 3. Average Percentage of Automatically Generated WSML

<table>
<thead>
<tr>
<th>WSML Component</th>
<th>Percentage of WSML Automatically Generated</th>
</tr>
</thead>
<tbody>
<tr>
<td>Web Service Description (Total)</td>
<td>76 %</td>
</tr>
<tr>
<td>Capability</td>
<td>47%</td>
</tr>
<tr>
<td>Interface</td>
<td>82%</td>
</tr>
<tr>
<td>Goal Description</td>
<td>74%</td>
</tr>
</tbody>
</table>

In addition to the ability to consistently generate the required non functional properties and appropriate keywords necessary to the structure of a WSML file, the transformation clearly fulfills a greater portion of the interface description than the other components of the description. The state signature is very well represented by the transformation and this is partially due to incorporating the annotation of grounding information in the feature model. Comparatively, if more information could be annotated into the feature model, presumably this automation percentage would increase; while if the grounding information was removed from the feature model, the percentage of automation would decrease. This suggests that the more information about the real-world Web services which can be annotated into the feature model at design time, the greater levels of automation are attainable. It also suggests that should the transformation be able to obtain and extract the semantic descriptions of the
molecular services, a great deal more automation could be attained. However, this is not possible in the current transformation environment and, in practice, when grounding choreography to the WSDL file of real-world services, there is no guarantee that they are backed by semantic descriptions in WSML. In fact, they may be described in OWL-S or some other description language, or have no semantic descriptions at all. A transformation, unless it can access a discovery engine itself, is not likely to perform this task. It is also expected that the percentage of automation would be even higher if orchestration was implemented so that the choreography description of the composite service would not have to provide so much of the necessary communication details of its molecular services. The ability to invoke molecular services through orchestration without requiring all the complex details of their internal choreography has the potential to properly preserve the division between choreography and orchestration with a resulting efficiency in automation.

The relatively low percentage of automation for the capability description is primarily due to the requirement to manually insert any shared variables and consider how these variables are reflected in the precondition of the first service in the composition and the postcondition of the last service in the composition. However, the number of words used in the capability is typically low compared to that of the interface, which includes the relatively wordy transition rules. Nonetheless, as the capability is essential in discovery, the manual intervention for this section is critical. Even if the transformation could extract more of this information automatically, it would likely be worthwhile for the developer to check the results of the transformation to ensure that discovery would occur as expected.

In terms of the Goal description, this is derived from the Web service description and therefore it shows a close correlation with the automation percentage of the Web service itself. Not represented in Table 3 for the Goal is the requirement for an instance ontology to provide sample input data for the execution to use. This is always expected to be handled through manual intervention and it is
reasonable to expect that user goals, where they specifically relate to instance data, should be manually entered by the developer for these purposes. It is unlikely that such instance data would be provided during feature modeling. It would likely be possible to automate such a process when the intention is to create many instance ontologies for rigorous unit testing of a specific composition, but this would most likely be done in conjunction with a full testing suite in a production environment. For the purposes of determining whether a proposed composition is possible or not, this would not be necessary.

**EASE OF USE.** The solution created in Chapter 4 and demonstrated in Chapter 5 is efficient, effective, and accurate. However, as expected, it is somewhat hindered by a requirement for manual intervention and there are some issues with tool support compatibility. Feature modeling itself is primarily a manual exercise and this methodology only adds a set of design rules to feature modeling which are relatively easy to follow. As the feature model .fmp file is.ecore-based, loading the file into the transformation is simply a matter of changing the file extension from .fmp to .ecore. Running the SOA2WSMO.atl transformation is easily done provided the Run configuration references the input and output metamodels (uri:http:///fmp.ecore in the EMF Registry and WSMO.ecore) as well as the input and output files. The output is a .ecore file and this must be renamed to a .wsml.xmi file in order to run the *Extract WSMO Model to WSML File* utility provided by the TCS plugin. It is at this point where we have compatibility issues. The versions of Eclipse that some of the plugins work within are different. At the present time, the Feature Plugin can only work in Eclipse 3.2. Unfortunately, the version of TCS which runs in Eclipse 3.2 was an alpha version and was found to not have all the functionality required for this work. Therefore, for feature modeling and ATL model transformation Eclipse 3.2 is used; for extracting the WSMO model to WSML language and subsequent execution in WSMX, Eclipse 3.4.1 is used. This requires a few extra steps (copy, paste, and rename of files) as well as having two different instances of the Eclipse development environment open. While not reducing the research value of this
work, it does suggest that more work would need to be done to produce a fully integrated and streamlined solution for this process to be practical in a production environment. However, considering feature modeling is primarily a manual task and the developer is likely going to spend more time on the other aspects of the overall work (such as creating and describing molecular Web services, modeling them, and deriving product configurations) it is not expected that this represents a significant barrier to adoption.

**SCALABILITY.** It is expected that this solution scales well in some aspects and less well in others. The scalability is, in part, determined by the available tool support associated with the various formalisms. The Feature Plugin can be used to model very large feature models. As such, provided the feature modeling practices proposed in this work are applied recursively, the transformation will perform as expected. However, extremely large feature models can be challenging to maintain, especially if they represent real-world services in a changing SOA environment. In terms of manual intervention, scalability is definitely an issue. The primary sources of manual intervention relate to the internal content of the molecular Web services. Should many Web services be composed into a single composition, this increases the work of the developer to satisfy the details of the capability and transition rules which come directly from molecular Web service descriptions. While no composite Web services were found in the literature which utilize a very large number of services, this is not beyond the realm of possibility in the future. It would therefore be important to find a means by which semantic descriptions could be located, and details extracted automatically, of the real world Web services employed as molecular services. As it is not necessary for every grounded Web service to have semantic descriptions in WSML to be used in a composition by WSMX (provided the composite Web service has in mode containers pointing to actual grounding), it is not currently possible to assume that this information can always be obtained on the fly. In fact, to do so would require full integration of
the transformation with a discovery engine. Also, should the choreography become more complex, and therefore different combinations of guarded transitions in the ASM required, there would likely be more manual intervention required to handle the different branching and looping necessary to represent the actual process. The transformation developed in this work does not exercise all possible transition rule structures available in the very expressive variant of WSML expected by WSMX and other current WSMO tools, WSML-Flight. Primarily, forall loop structures are assumed in this work and this is consistent with WSMO examples in the literature. Rule structures such as choose and uncond (with piped rules) are not expected or handled by this transformation. While there are many ways to represent business processes in WSML transition rules, transition rule constructs beyond the basic rules and not required to implement the solutions of the running case or examples in the literature are outside the scope of this work.

Another scalability issue is related to the execution environment itself, WSMX. All Web services which pass data to and from the execution environment in a choreography require lifting and lowering adapters, as described in Chapter 4. This means that WSMX itself requires adapters which implement the ie.deri.wsmx.adapter(Adapter interface. In a full production environment (i.e. not using the test framework provided by WSMX) this requires a rebuild of the invoker component from WSMX source every time a Web service requiring a different adapter is included in a configuration. In practice, this means that every Web service added will involve some addition of an adapter or modification of an existing adapter, unless very strict standards and conventions are followed in creating each subsequent Web service. In the heterogeneous world of SOA, this is not likely to be enforceable and therefore the scalability limits of the current version of WSMX are inherited by this work.
A final aspect of scalability involves the likelihood of the specification or implementations of WSML evolving. While employing MDD and focusing transformation on the metamodel ensures an easier to maintain solution, there is no guarantee that as the WSMO specification evolves and the implementations improve that this transformation process would not require some change. First, should orchestration be implemented fully in specification and tool support (a welcome improvement to the current state of the art) this would require changes to the transformation and possibly even the metamodel and textual concrete syntax artifacts used to support the transformation. As the WSMX execution environment develops more functionality, the requirements for manual intervention might increase should certain new NFPs be employed. Already there are NFPs which relate to specific discovery components and the use of lifting and lowering adapters. If these NFP requirements diverge between different WSMO implementations, there may be issues where the transformation is producing output which cannot be executed or otherwise utilized in differing WSMO tool support.

**PERFORMANCE.** The transformation of the feature model to the output WSML takes mere seconds with the size of feature model used in this running case. For the demonstration transformation with the running case input, the time for ATL transformation was 2.948s and the number of bytecode instructions was 861532. Considering other runs of the transformation, in no instance was the execution time for the ATL transformation greater than 3.5 seconds. Similarly, the extraction of the WSML from the output WSMO using TCS is also relatively fast. Comparatively, the developer would typically take far more time copying and renaming the file from one Eclipse version to the next. Typically, successful execution of the running case composition in WSMX takes approximately 4000 ms and is often less. Considering the time spent on other activities in this type of modeling exercise, this is a trivial delay in getting information to the developer. It should be noted that all Web services were deployed through localhost and it could be expected that invoking any molecular services out on the actual Internet would
add time to the execution. Far more relevant to the developer is the time spent evaluating the WSMX console output when discovery, data mediation, or process mediation is not achieved. Clear notification of successful composition execution comes fairly quickly, as does notification of such failures as failure to parse, lack of discovery, and inconsistent ontologies. However, some errors related to process mediation involve evaluating fairly verbose console output. While not directly related to the actual performance of this solution it needs to be taken into account in the overall usefulness to the developer. Where certain heuristics can be applied to common transition rule errors, it may be possible at some point to add logging or notification functionality to WSMX components to explicitly signify such problems when they occur. In a production environment, this would be a worthwhile improvement to this solution. However, it is beyond the scope of this work.

6.3 COMPARISON

WSMO STUDIO AND WSMT. WSMX is not the only implementation of the WSMO specification. However, being a full execution environment it certainly is capable of a wider range of reasoning functionalities than some of the other tools. On the other hand, the other implementations of WSMO are still worthy of consideration and comparison. Two commonly used utilities are WSMO Studio and WSMT. WSMT was directly utilized in the detailed design for the creation of semantic descriptions for the molecular services and was also utilized for manual intervention. These descriptions and the composite service descriptions were also imported into WSMO Studio to evaluate that tool as well. Both WSMT and WSMO Studio employ reasoners, and even discovery engines, to provide information and accomplish tasks. Both WSMT and WSMO Studio can use reasoners, such as IRIS, MINS, or Pellet (the user can usually choose) to determine whether the process and domain ontologies used are satisfyable. The reasoners can also be used to manually conduct queries of these ontologies. However, neither facilitate any significant reasoning on the actual semantic descriptions of
the Web services. While WSMT can simulate both keyword-based and lightweight discovery with goals and services in the local repository, WSMO Studio only provides a Quality of Service (QoS) based discovery engine which would not be suitable for the purposes of this work. While neither of the tools can even parse orchestration descriptions, WSMO Studio also does not validate choreography descriptions. Overall, these tools are suitable for manual development and preliminary testing of semantic descriptions of Web services and goals, as well as ontologies. However, no significant reasoning is possible beyond the process and domain ontologies utilized by the Web services. They can provide no insight into the capability or interface of a composite Web service description.

**COMPARISONS WITH OTHER WORK.** There are a number of other designs which pursue similar goals to this work. As is clear from the literature review of Chapter 2, there are only limited mature research efforts which seek to integrate SPL and SOA formalisms. Nonetheless, there are a number of highly insightful solutions which develop transformations with one or both of the formalisms used in this work. Important similarities and differences are considered in this section.

This work builds on the progress demonstrated by Wang *et al.* (2007) but also extends the reasoning approach in a different direction. Their work is primarily focused on using ontology reasoning to verify feature models and product configurations. While their work is successful at accomplishing this, it is applied only to OWL ontologies. It does not specifically address feature models composed of Web services and therefore makes no mention of what could potentially be done with OWL-S. In the context of this work's running case, where real-world services are proposed for composition, their approach, without extension, would not return much reasoning value beyond what the feature modeling environment already provides. They note that the reasoners employed do not provide information as to why classes may be found to be inconsistent. Their work provides some initial progress in making available debugging aids to this verification process. However, in the design
of this work, the Feature Plugin already ensures that product configurations are consistent with their associated feature model. There is therefore no need to verify product configurations in that way in our particular environment. The real value is in determining how the product configurations can actually work with the services they represent. This requires some reasoning beyond that possible in the feature model structure alone and yet also beyond that provided by any domain or process ontology alone. This design instead utilizes an actual execution environment (which employs the reasoners) and a testing framework to integrate the two formalisms and provide more precise debugging aids specifically in the context of Web service composition. With this design, inconsistencies with product configurations can be found beyond those possible in their approach. Nonetheless, their work, although utilizing OWL and not applied to feature models comprised of Web services, relates favourably to the overall goal of applying ontology-based reasoning to the design of software product lines. The authors note specifically that more expressive languages may facilitate more complex relationships to reason against. As choreography and orchestration can represent fairly complex processes, and WSML is more expressive than OWL, there are possibilities for further application of their work into this domain. This work serves as an early contribution in that regard.

The work of Brambilla et al. (2007) provides significant guidance to this work and serves as a notable comparison. While their work is utilizing the much more expressive BPMN as the base from which to extract WSML descriptions, the principles are similar. Like this work, the authors use MDD to extract significant portions of executable WSML descriptions from other modeling artifacts. While BPMN is more expressive and naturally represents business processes, as opposed to coercing the feature model to incorporate business process information as done in this work, BPMN does not allow for the modeling of variability the way that feature models can. Therefore, without extension, it is of limited use in the application of SPL methods. However, their percentages of automation from model
to WSML are quite high, as should be expected with more expressive input. They also derive orchestration descriptions although they note the limited utility these currently have. Given their more expressive input, they are also able to extract WSMO mediators and specific WSMO goals. This is a significant benefit that this work does not achieve. The authors are also able to extract portions of domain and process ontologies from the other modeling artifacts. This work utilizes existing domain and process ontologies so this extension is not necessary here. They have found, as did this work, that choreography typically requires significant annotation by the designer to account for all the possible interaction sequences. Like this work, they do carry the process beyond simply creating output by integrating into a working case with discovery and mediation. They utilize the WSMO-compliant discovery engine, Glue, while this is not available through the WSMX environment that this work uses. The use of Glue necessitates the development of WSMO mediators and therefore this provides significant value. Overall, their work compares favourably with this work in terms of productivity, ease of use, scalability, and though not addressed in detail, likely performance as well. Their exploitation of semantic WSMO descriptions validates this work’s selection of WSMO as the target formalism for the SOA output. Where this work adds value beyond that of Brambilla et al. (2007) is in supporting the reuse and modeling benefits possible with the feature model. Though less expressive than BPMN, it does provide benefits that cannot be attained by BPMN alone. It is conceivable that feature models could be derived, with some manual intervention, as an interim artifact during their process. It would represent an interesting convergence between the two approaches but is clearly beyond the scope of their work.

Dodero et al. (2007) integrate feature models and WSMO in their work in the innovative development of automated instructional design. WSMO Goals translate into product configurations of a feature model while WSMO ontologies are used to manage additional constraints which are not
possible with standard feature modeling. Conceptually, their work provides a similar benefit as this work does, where constraints not possibly incorporated directly within the feature model formalism are nonetheless made available to developers through WSMO ontology and Web service descriptions. WSMO Studio is used to create WSMO Ontologies and Goals and, while these elements are related to the feature model, there is no description of an automated transformation between the two formalisms. For the most part, the two formalisms are connected to each other but not fully integrated. Although product configurations map to WSMO Goals, the feature selections within these configurations do not explicitly refer to any WSMO ontology concepts or services, merely qualities of the learning object. The primary use for a WSMO Goal is to facilitate discovery of WSMO Web Services. This usage is not reflected in their work. Therefore, the implementation of any advanced functionality facilitated by WSMO, such as discovery, invocation, and composition is not yet considered. However, despite this apparent underutilization of WSMO, their work nonetheless is able to provide some ontology reasoning to the feature modeling perspective and therefore demonstrates an interesting alternate approach to this work.

Montero et al. (2008) extend the meaning of feature models to be able to represent the basic structure of business processes by developing a set of mapping rules from feature models to BPMN. This work adopts their basic mapping rules for the purposes of interpreting the feature model. Both their work and this work establish that the order of features in the feature model can be interpreted for the purposes of business process. This is an extension of the meaning of feature models although without extending the basic notation of the feature model structure itself. Similarly, like this work, their process also automatically generates business process information for implementation into an execution engine. In their case, it was automatic generation into BPMN, which has a formal mapping into an execution language, BPEL. In this work though, the transformation is more direct. There was no need
for an intermediate formalism such as BPMN, as this work was able to map straight to executable WSML Web service descriptions. Furthermore, while their methodology does result in executable BPEL, this language is syntactic and not semantic. Semantic reasoning is therefore not supported by their approach. Neither this work nor theirs was free from the requirement for manual intervention. In both cases, some manual intervention was needed to ensure completely executable output. While their work provides an important basis upon which to interpret additional meaning from the feature model, this work extends their approach to incorporate semantic information into the business process through direct transformation into WSMO.

The related work of Kaviani et al. (2008) and Mohabbati et al. (2009) utilize ontologies for the annotation of feature models. This approach incorporates semantic information into the feature model to facilitate greater reasoning. With a focus on incorporating essential non-functional requirements into the commonality and variability relationships that feature models represent, their work allows for advanced reasoning to interpret relations between features that are not possible to be obtained from the feature model structure alone. Similar to this work, their approach provides important real-world validation regarding the product configuration from the feature model. Their work builds on the mapping from feature models to OWL provided by Wang et al. (2007) while this work maps to WSMO instead. This therefore necessitates the transformation of the entire feature model as well as the product configuration while this work transforms only the product configuration as the focus is primarily on reasoning about the business process itself and not any of the non-functional properties. While this work does provide some handling of important non-functional properties which are required by the WSMX execution environment, this makes limited use of the reasoning which could be considered from the non-functional properties associated with the services represented as features in the feature model. Combining both approaches, as proposed by Mohabbati et al. (2009) has the potential to
provide significantly more information to the developer when composing product configurations from feature models of services.


CHAPTER 7
CONCLUSIONS AND RECOMMENDATIONS

7.1 CONCLUSIONS

This work serves as a proof of concept demonstrating that it is possible to integrate the two formalisms, feature models and the Web Service Modeling Ontology, to facilitate Semantic Web-based reasoning in the design of software product lines, specifically those comprised of Web services. The objective was to utilize the feature model formalism to organize Web services into a software product line and then export resulting product configurations into WSMO for the dual purposes of having usable executable output and extracting semantic reasoning based information to assist in domain and application engineering. The design of a hypothetical, yet realistic, running case lead to the creation of a set of test case Web services to test the process. These Web services were organized into a feature model in a manner which reflects the overall business logic as well as representing the variability inherent in the feature model formalism. These feature modeling rules were fully described and demonstrated. Upon creation of a product configuration from the feature model, the resulting product configuration was transformed into an executable composite Web service description in WSML, though some manual intervention was required. This composite Web service was loaded into a test environment supported by the WSMX execution environment to simulate the actual discovery, data mediation, and process mediation of the proposed composition. These reasoning-driven processes were shown to provide significant feedback to the developer in considering the selected features as a composition. Table 4 lists the final artifact deliverables which comprise the design of this solution.
Table 4. Summary of Design Deliverables

<table>
<thead>
<tr>
<th>Filename</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SOA2WSMO.atl</td>
<td>Transformation from Feature Model to WSML Web Service Description</td>
</tr>
<tr>
<td>WSMO.km3</td>
<td>WSMO Metamodel Used in Above Transformation</td>
</tr>
<tr>
<td>WSMO.tcs</td>
<td>Textual Concrete Syntax for WSML</td>
</tr>
<tr>
<td>Comp699Adapter.java</td>
<td>Lifting and Lowering Adapter Implementing WSMX Adapter interface</td>
</tr>
<tr>
<td>Invoker.java</td>
<td>Modified WSMX Source to Control Use of Custom Adapter Above</td>
</tr>
</tbody>
</table>

This work has contributed an MDD-based ATL transformation between the feature model formalism and the Web Service Modeling Ontology formalism, represented in the file SOA2WSMO.atl. In order to achieve successful transformation, this work also developed a KM3 metamodel of the WSML language (specifically the WSML-Flight variant) in the WSMO.km3 file (although unused WSMO elements, such as Mediators, were not implemented). This, in turn, required the definition of a textual concrete syntax for WSML, WSMO.tcs. These two files can be utilized as is, or extended, for other MDD purposes involving the WSMO language. Finally, modifications to the WSMX source code were necessary in this work, specifically modifications to the Invoker component, invoker.java, and the creation of a custom lifting/lowering adapter, Comp699Adapter.java. For any users of WSMX requiring examples of extending the WSMX execution environment for their own custom SOA, these serve as potential guidance.

In terms of significance of this work, Semantic Web-based reasoning is applied against the constraints which lie outside of the immediate perspective of the feature model while retaining the SPL-related advantages of modeling with feature models. The feature model is an important and popular formalism to model SPL. However, there are short-comings in this formalism, especially when considering the modeling of Web services as features. The utilization of the solutions in this work allow the developer to preserve the feature model perspective and yet obtain detailed and specific reasoning-based information regarding how the product configurations function with the actual services.
in terms of discovery, invocation, and composition. This information simply cannot be perceived from the basic feature model perspective and yet is critical information to the developer. A product configuration may be consistent with its associated feature model but the actual constraints in terms of Web service capability and interface are too complex to incorporate into the feature model. This solution extends the reasoning which can be done while feature modeling to include the real-world constraints of the individual Web services.

7.2 RECOMMENDATIONS

This approach is useful for developers attempting to create Software Product Lines out of Web services. This work facilitates greater reuse for SOA by utilizing both SPL practices and Semantic Web-based reasoning. The advantages of doing this have been identified in this work, as well as the proof of concept that this is possible to achieve. The principles of navigation of the feature model through an MDD transformation can be used as defined in this work or extended to customize the output of the transformation. In the case of additional annotation to feature models, or extension of the basic feature model, the transformation could be extended to accommodate and make use of the additional information. An example of this with respect to service grounding information was demonstrated. The Semantic Web-based reasoning information which has been demonstrated is clearly applicable to the application engineering (i.e. composition) of these models. While the feature model environment is suitable enough for ensuring product configurations are validated against their respective feature models, the developer can incorporate the process described here to obtain valuable information about how the composition could function in the real-world environment. Other deliverables, such as the metamodel and textual concrete syntax for WSML, can also be utilized for any number of other purposes.
There is an active research community working to integrate both SPL and SOA approaches. The ongoing research centered on the integrated development of Service Oriented Architectures and Software Product Lines (SOAPL) is both a driver of this particular work and a potential user of its findings. Workshops related to this integrated set of practices and tools have been formally a part of Software Product Lines research since the initial workshop report published by Cohen and Krut (2008) with individual contributions occurring much earlier, as discussed in Chapter 2. This work is an improvement on an earlier design (Rusk and Gašević 2008) and the further refinement of it has the potential to contribute more to the overall goals of this emerging field. Already, we see how this work is applicable to that proposed by Mohabbati et al. (2009) and there are other possibilities to refine this work to contribute to other work. The approaches to service-oriented product line architectures are improving and further meaning from feature models has been demonstrated. The granularity solutions demonstrated by Lee et al. (2008) have been refined into the layered architectural style proposed by Medeiros et al. (2009). While those two proposes do not yet fully account for business process within the variability, it is likely that this work will provide a useful basis to further research which may propose to do so exactly that.

7.3 SUGGESTIONS FOR FURTHER RESEARCH

Within the immediate domain of this work, there is certainly opportunity to improve and streamline the process as the associated tool support and related specifications evolve. The WSMO specification is still not finalized and the WSML language does not yet completely implement all aspects of the WSMO framework. Foremost among these is that of orchestration. Orchestration is well-suited to representing the internal process of a Web service composition. In fact, that is its intended role. The necessity of using choreography to achieve this is not ideal. Once complete orchestration descriptions can be reasoned against in an execution environment, and the transformation modified
accordingly, there can likely be improved automation and greater reasoning achieved. Related to this immediate domain but again somewhat dependent on advances elsewhere is that of the annotation of feature modeling. It has been shown in this work that the more information which can be included in the feature model formalism, the greater the automation attainable. As enhancements to the basic feature model metamodel are incorporated, the transformation could be updated to achieve improved automation and greater reasoning. Further research with the potential for immediate benefits is therefore that of improving the tools relating to WSMO Web services and feature modeling. Few of these tools are able to be used in production environments without significant improvement and their relative immaturity certainly limits what can be accomplished in the direction of this work.

As greater meaning with respect to SOA is assigned to feature models, there is the likelihood that there will be further progress towards integration of the different approaches. The architectural styles used to represent services in feature models, such as those demonstrated by Lee et al. (2008) and Medeiros et al. (2009), do not yet inherently model either the orchestration or choreography of services. Therefore, MDD is not yet possible to be utilized in automating the progress from architecture specification to actual executable artifacts which can serve in a production environment. There is still a sizable gap between SOA and SPL approaches. However, this work and the many works cited here are rapidly closing that gap and leading to solutions which are applicable to real-world production environments.

Finally, as mentioned in Chapter 6, pairing the transformation with discovery engine functionality could automate even the extraction of complete molecular Web service descriptions. This has the potential to entirely automate the process though it would take a significant extension to the transformation. With simple compositions composed of Web services within known repositories, as in this running case, this is not likely worth the effort. However, to achieve truly autonomous
configuration, the principles of such a design have far-reaching implications. It would allow the feature modeling of goals, instead of services, where the results of product configurations are provided to a discovery engine to find services which meet the goals. This combination of design time feature modeling with run time discovery facilitates feature modeling of abstract requirements instead of known Web services, greatly expanding the range of options for composition. However, the scalability issues already identified in this work would be exacerbated when pushed to the runtime environment. Nonetheless, designing a feature model to WSMO transformation which could conduct its own discovery and other advanced reasoning tasks on semantically described real-world services brings the state of the art much closer to a true Semantic Web.
REFERENCES


In Proceedings of the 2004 OOPSLA Workshop on Eclipse Technology eXchange. (pp. 67-72).
Vancouver, BC, Canada: Association for Computing Machinery.


and feature interaction detection using critical pair analysis. In Model Driven Engineering Languages and Systems. (pp. 151-165). Berlin, Germany: Springer.


Companion to the 19th Annual ACM SIGPLAN Conference on Object-Oriented Programming
 Systems, Languages, and Applications OOPSLA ’04. (pp. 18-19). Vancouver, BC, Canada:
 Association for Computing Machinery.


 the ACM 49, 12, 29-32.

 In Proceedings of the 10th IEEE International Conference on Engineering of Complex
 Computer Systems. (pp. 303-312). IEEE.

 composition of Semantic Web services. Web Semantics: Science, Services and Agents on the
 World Wide Web 1, 27-46.


selection for mobile devices. In 11th International Software Product Line Conference. (pp. 129-140). Kyoto, Japan: IEEE.

APPENDIX A

ATL TRANSFORMATION: FM2WSMO.atl

This file is available for download at:

http://io.acad.athabascau.ca/~jeffr/soa2wsmo.atl
APPENDIX B

WSMO KM3 METAMODEL FILE: WSMO.ecore

This file is available for download at:

http://io.acad.athabascau.ca/~jeffr/wsmo.km3
APPENDIX C

TCS TEXTUAL CONCRETE SYNTAX FOR WSML: WSMO.tcs

This file is available for download at:

http://io.acad.athabascau.ca/~jeffr/wsmo.tcs
APPENDIX D

ADAPTER FOR LIFTING SOAP-BASED XML AND LOWERING WSML

This file is available for download at:

http://io.acad.athabascau.ca/~jeffr/Comp699Adapter.java
APPENDIX E

JAVA ARTIFACTS FOR DEMONSTRATION WEB SERVICES

This zipped package of files is available at:

http://io.acad.athabascau.ca/~jeffr/ WsmoJavaFiles.zip
APPENDIX F
SEMANTIC WEB SERVICE DESCRIPTIONS AND RELATED ONTOLOGIES

This zipped package of files is available at:

http://io.acad.athabascau.ca/~jeffr/WSML-Test-Files.zip