A Run-Time Approach of Combining Ontologies to Enhance Interactive Requirements Elicitation for Software Customization

Shubhrendu Tripathi
University of Windsor

Follow this and additional works at: https://scholar.uwindsor.ca/etd

Recommended Citation
Tripathi, Shubhrendu, "A Run-Time Approach of Combining Ontologies to Enhance Interactive Requirements Elicitation for Software Customization" (2016). Electronic Theses and Dissertations. 5778.
https://scholar.uwindsor.ca/etd/5778

This online database contains the full-text of PhD dissertations and Masters' theses of University of Windsor students from 1954 forward. These documents are made available for personal study and research purposes only, in accordance with the Canadian Copyright Act and the Creative Commons license—CC BY-NC-ND (Attribution, Non-Commercial, No Derivative Works). Under this license, works must always be attributed to the copyright holder (original author), cannot be used for any commercial purposes, and may not be altered. Any other use would require the permission of the copyright holder. Students may inquire about withdrawing their dissertation and/or thesis from this database. For additional inquiries, please contact the repository administrator via email (scholarship@uwindsor.ca) or by telephone at 519-253-3000 ext. 3208.
A Run-Time Approach of Combining Ontologies to Enhance Interactive Requirements Elicitation for Software Customization

by

Shubhrendu Tripathi

A Thesis
Submitted to the Faculty of Graduate Studies through Computer Science in Partial Fulfillment of the Requirements for the Degree of Master of Science at the University of Windsor

Windsor, Ontario, Canada
2016

© 2016 Shubhrendu Tripathi
A Run-Time Approach of Combining Ontologies to Enhance Interactive Requirements Elicitation for Software Customization

by

Shubhrendu Tripathi

APPROVED BY:

______________________________
Dr. A. Azab
Dept. of Industrial & Manufacturing Systems Engineering

______________________________
Dr. S. Goodwin
School of Computer Science

______________________________
Dr. X. Yuan, Advisor
School of Computer Science

May 20, 2016
Declaration of Co-Authorship / Previous Publication

I. Co-Authorship Declaration

I hereby declare that this thesis incorporates material that is result of joint research of the author and his supervisor Prof. Xiaobu Yuan. This joint research has been published / submitted to various conferences that are listed below.

I am aware of the University of Windsor Senate Policy on Authorship and I certify that I have properly acknowledged the contribution of other researchers to my thesis, and have obtained written permission from Prof. Xiaobu Yuan to include those materials in my thesis.

I certify that, with the above qualification, this thesis, and the research to which it refers, is the product of my own work.
II. Declaration of Previous Publication

This thesis includes two original papers that have been previously published / submitted for publication in peer reviewed conferences, as follows:

<table>
<thead>
<tr>
<th>Thesis Chapter</th>
<th>Publication title / Full citation</th>
<th>Publication status</th>
</tr>
</thead>
<tbody>
<tr>
<td>Chapter 4,</td>
<td></td>
<td>(Published)</td>
</tr>
<tr>
<td>Chapter 5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Chapter 4,</td>
<td></td>
<td>(In press)</td>
</tr>
<tr>
<td>Chapter 5</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

I certify that I have obtained written permissions from the copyright owners to include the above published materials in my thesis. I certify that the above material describes work completed during my registration as graduate student at the University of Windsor.

I declare that, to the best of my knowledge, my thesis does not infringe upon anyone’s copyright nor violate any proprietary rights and that any ideas, techniques, quotations, or any other material from the work of other people included in my thesis, published or otherwise, are fully acknowledged in accordance with the standard referencing practices. Furthermore, to the extent that I have included copyrighted material that surpasses the bounds of fair dealing within the meaning of the Canada Copyright Act, I certify that the copyright owners of the two papers, referenced above, allow (IEEE)
copyrighted material to be used in this thesis without additional written permission.  

I declare that this is a true copy of my thesis, including any final revisions, as approved by my thesis committee and the Graduate Studies office, and that this thesis has not been submitted for a higher degree to any other University or Institution.
Abstract

This thesis highlights the recent developments in Requirements Engineering for Software Product Line Engineering, with a focus on the use of ontology in interactive Requirements Elicitation and the existing techniques of ontology operations. Recent research done in Requirements Elicitation has been towards using ontologies as a modeling basis for gathering requirements. A new algorithm has been developed to allow ontologies to be combined at run-time when gathering the requirements of software clients. By harnessing knowledge in other ontologies, a more refined set of requirements can be generated. A scenario illustrating the use of ontology combination towards acquiring requirements for mobile platforms is also provided. The proposed method further enhances the capability of interactive software customization, thus helping to make Software Product Line Engineering a new practice in software development.
Dedication

This thesis is dedicated to my dear mother, sister, and late father for their endless love and support.
Acknowledgements

I would like to sincerely thank and express my deep sense of gratitude for my supervisor, Dr. Yuan. He has been very kind and patient throughout this endeavour. Without his clear and effective guidance, this work would not have been possible. I have been very fortunate to have a supervisor with a great intellect and vast knowledge, who cared so much about my work, and who responded to my questions and queries so promptly. He was consistently supportive and constantly encouraged me to improve and refine my work. He was always available to discuss any issues, big or small, I had during my research.

I also want to thank Dr. Goodwin and Dr. Azab. Their comments and suggestions have been invaluable and have greatly improved the quality of this work.
1 Introduction

2 Related Work

2.1 Overview

2.2 Software Customization

2.2.1 Overview

2.2.2 Software Product Line Engineering

2.2.3 Service Oriented Architecture

2.2.4 Integrating SPL and SOA

2.3 Requirements Elicitation

2.3.1 Overview
TABLE OF CONTENTS

2.3.2 Requirements Elicitation .............................................. 7
2.3.3 Ontologies in RE ....................................................... 9
2.4 Interactive Requirements Elicitation .................................. 11
2.5 Ontology and Operations ................................................. 14
  2.5.1 Overview .......................................................... 14
  2.5.2 What is an Ontology? ................................................ 14
  2.5.3 Operations on Ontologies .......................................... 16
    2.5.3.1 Matching ...................................................... 16
    2.5.3.2 Alignment .................................................... 18
    2.5.3.3 Mapping ...................................................... 19
    2.5.3.4 Integration ................................................... 20
    2.5.3.5 Merging ....................................................... 21
3 A New Method of Ontology Combination .................................. 25
  3.1 Overview .......................................................... 25
  3.2 Problem Statement ..................................................... 25
  3.3 Ontology Combination ................................................. 26
  3.4 Example .......................................................... 27
  3.5 Proposed Methodology ................................................ 31
    3.5.1 Definitions .................................................. 31
    3.5.2 Step 1: Generate Correspondences ......................... 32
    3.5.3 Step 2: Generate Relationships ......................... 32
    3.5.4 Step 3: Check consistency of combined ontology, \( O_c \) ... 32
    3.5.5 Step 4: Validation of \( O_c \) .................................... 32
  3.6 Design of Algorithms ................................................ 35
    3.6.1 Overview .................................................. 35
    3.6.2 SelectLink algorithm ........................................ 36
    3.6.3 GetCorrespondences algorithm .......................... 37
    3.6.4 GetRelationship algorithm ................................. 40
    3.6.5 FindRelationshipJWNL algorithm ....................... 43
<table>
<thead>
<tr>
<th>3.6.6 GetHighestCM algorithm</th>
<th>44</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.6.7 Time Complexity</td>
<td>45</td>
</tr>
</tbody>
</table>

4 Experiments

<table>
<thead>
<tr>
<th>4.1 Overview</th>
<th>47</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.2 Software</td>
<td>47</td>
</tr>
<tr>
<td>4.3 Interface</td>
<td>48</td>
</tr>
<tr>
<td>4.4 Experiments</td>
<td>51</td>
</tr>
<tr>
<td>4.4.1 Scenario I - Single Ontology</td>
<td>51</td>
</tr>
<tr>
<td>4.4.2 Scenario II - Multiple Ontologies</td>
<td>51</td>
</tr>
<tr>
<td>4.4.3 Scenario III - Ontology of Mobile SOA Functions</td>
<td>60</td>
</tr>
<tr>
<td>4.5 Case Study</td>
<td>63</td>
</tr>
<tr>
<td>4.6 Contributions</td>
<td>76</td>
</tr>
<tr>
<td>4.6.1 Overview</td>
<td>76</td>
</tr>
<tr>
<td>4.6.2 Enhanced Interactive Requirements Elicitation</td>
<td>76</td>
</tr>
<tr>
<td>4.6.3 Extending Customization to Mobile Applications</td>
<td>77</td>
</tr>
</tbody>
</table>

5 Conclusion and Future Directions

<table>
<thead>
<tr>
<th>5.1 Overview</th>
<th>78</th>
</tr>
</thead>
<tbody>
<tr>
<td>5.1.1 Conclusion</td>
<td>78</td>
</tr>
<tr>
<td>5.1.2 Future Directions</td>
<td>79</td>
</tr>
</tbody>
</table>

Bibliography

Vita Auctoris
List of Tables

2.1 Summary of Ontologies in RE ........................................... 10
2.2 Comparison of Ontology Operations ................................. 24
3.1 WordNet and Relationships ........................................... 42
3.2 Time Complexity of Combine algorithm ........................... 46
List of Figures

2.1 Software Product Line Engineering [59] ............................................. 4
2.2 Service Oriented Architecture [22] .................................................... 5
2.3 Requirements Engineering processes [68] ......................................... 8
2.4 Ontology-based Requirement Model [73] .......................................... 11
2.5 Pseudo code for requirement evaluation process [73] .......................... 12
2.6 Interactive Requirements Elicitation system [73] ................................. 13
2.7 Ontology as a Semantic Network [27] ................................................. 14
2.8 Ontology as a UML model [27] .......................................................... 15
2.9 Ontology represented in OWL (excerpt) [27] ...................................... 15
2.10 Ontology Matching [42] .................................................................... 17
2.11 Partial view of an Ontology Alignment [25] ....................................... 18
2.12 Ontology Mapping ............................................................................. 19
2.13 Ontology Integration [50] ................................................................. 20
2.14 Ontology Merging [39] ..................................................................... 22

3.1 Pizza ontology [10] ........................................................................... 28
3.2 Food ontology [4] ............................................................................... 29
3.3 Excerpt of the Combined Ontology .................................................... 30
3.4 Methodology ....................................................................................... 34

4.1 Existing interface of Interactive Requirements Elicitation system ......... 48
4.2 Ontology Combination Viewer (OC Viewer) ....................................... 49
4.3 Initial state of Interactive Requirements Elicitation system ................. 49
4.4 State of Interactive Requirements Elicitation system after selecting some requirements .................................. 50
4.5 BookStore ontology, $O_{SPL}$ in $K_{RE}$ ......................................................... 52
4.6 Search ontology, $O_i$ in $K_{RE}$ ............................................................. 53
4.7 OrderSummary ontology, $O_j$ in $K_{RE}$ .................................................. 53
4.8 ManagePaymentInfo ontology, $O_k$ in $K_{RE}$ ........................................ 54
4.9 Excerpt of Combined Ontology - $O_{SPL}$ and $O_i$ .................................. 55
4.10 OC Viewer output after $O_{SPL}$ and $O_i$ combination ........................... 55
4.11 Excerpt of Combined Ontology - $O_{SPL}$ and $O_j$ ............................... 56
4.12 OC Viewer output after $O_{SPL}$ and $O_j$ combination ........................... 56
4.13 Excerpt of Combined Ontology - $O_{SPL}$ and $O_k$ ............................... 57
4.14 OC Viewer output after $O_{SPL}$ and $O_k$ combination ........................... 57
4.15 Complete Combined Ontology after three iterations ............................. 58
4.16 PlatformMobile ontology, $O_{Mobile}$ in $K_{RE}$ ....................................... 61
4.17 Excerpt of Combined Ontology - $O_{SPL}$ and $O_{Mobile}$ ...................... 62
4.18 OC Viewer output after $O_{SPL}$ and $O_{Mobile}$ combination .................. 62
4.19 Ontology Combination - Abstract view .............................................. 76
## List of Acronyms

<table>
<thead>
<tr>
<th>Acronym</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DL</td>
<td>Description Logic</td>
</tr>
<tr>
<td>JWNL</td>
<td>Java WordNet Library</td>
</tr>
<tr>
<td>OWL</td>
<td>Web Ontology Language</td>
</tr>
<tr>
<td>RE</td>
<td>Requirements Elicitation</td>
</tr>
<tr>
<td>SOA</td>
<td>Service Oriented Architecture</td>
</tr>
<tr>
<td>SPL</td>
<td>Software Product Line</td>
</tr>
<tr>
<td>SPLE</td>
<td>Software Product Line Engineering</td>
</tr>
</tbody>
</table>
## List of Listings

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.1 Dialogue Utterances (Part 1)</td>
<td>65</td>
</tr>
<tr>
<td>4.2 Dialogue Utterances (Part 2)</td>
<td>65</td>
</tr>
<tr>
<td>4.3 Dialogue Utterances (Part 3)</td>
<td>66</td>
</tr>
<tr>
<td>4.4 Dialogue Utterances (Part 4)</td>
<td>66</td>
</tr>
<tr>
<td>4.5 Dialogue Utterances (Part 5)</td>
<td>67</td>
</tr>
<tr>
<td>4.6 Dialogue Utterances (Part 6)</td>
<td>67</td>
</tr>
<tr>
<td>4.7 Dialogue Utterances (Part 7)</td>
<td>68</td>
</tr>
<tr>
<td>4.8 Dialogue Utterances (Part 8)</td>
<td>68</td>
</tr>
<tr>
<td>4.9 Dialogue Utterances (Part 9)</td>
<td>69</td>
</tr>
<tr>
<td>4.10 Dialogue Utterances (Part 10)</td>
<td>69</td>
</tr>
<tr>
<td>4.11 Dialogue Utterances (Part 11)</td>
<td>70</td>
</tr>
<tr>
<td>4.12 Picked Requirements</td>
<td>71</td>
</tr>
<tr>
<td>4.13 Abandoned Requirements</td>
<td>71</td>
</tr>
<tr>
<td>4.14 Entire OC Viewer output</td>
<td>72</td>
</tr>
<tr>
<td>4.15 OWL-S output file with platform-dependent details highlighted in gray</td>
<td>75</td>
</tr>
</tbody>
</table>
Chapter 1

Introduction

Software Product Line Engineering (SPLE) is an active area in Software Engineering. It holds the promise of making software customization as successful as the assembly-line process in the automotive industry. By reducing bloat of unwanted code in software systems, customization increases efficiency. In the near future of mobile, wearable and embedded devices [58], the size of a software program takes on an important dimension. By utilizing customization, software modules can potentially be assembled and re-assembled quickly to target different platforms in a cost-effective manner. Considerable progress [51] has been made in recent years for realizing this paradigm of software development.

One of the main subdisciplines of Software Engineering, including SPLE, is Requirements Engineering. Requirements Elicitation (RE) forms an important part of the Requirements Engineering process. A lot of effort [28] [60] has been put in this area of research. From the early 1990s to the present, many techniques have been identified to reduce errors and make the elicitation process work more efficiently. Ontologies have been used to try and ensure that RE is accomplished in a well-defined manner which in turn, ensures a robust implementation of a software system. Considerable progress has been made towards an interactive mode of RE for software customization [74]. Ontologies have been utilized for providing the foundation for such interactive systems.
They provide an excellent basis for representing concepts and the relationships between them. Due to this, they are being increasingly used across a variety of domains [45] [40].

The existing approach to interactive RE relies on using single ontologies to guide the interaction [73]. It would be more beneficial to harness knowledge available across multiple domains to dramatically improve the scope of interaction. Various operations on ontologies, such as merging, are design time operations and are thus not useful for an interactive system. Recently [73] with dialogue-based RE, an interactive way of gathering requirements has been made possible. This thesis proposes a novel method of Ontology Combinations. It is an approach to obtain knowledge in different ontologies when requirements elicitation is actually performed. The existing interactive approach uses a single ontology to drive the RE process of gathering requirements for building a customized Software Product Line (SPL) application. By bringing together different ontologies at run-time, this methodology promises to strengthen the interactive RE process and enhance it considerably. By combining multiple ontologies dynamically at run-time, a more detailed set of requirements can be obtained. This work defines the methodology for performing ontology combinations and presents a combine algorithm along with scenarios illustrating the approach. The contributions of this thesis are:

- An enhanced interactive RE process in which significantly more requirements are acquired from multiple ontologies through ontology combinations.

- Addition of mobile platform-dependent features to a customized SPL application by the use of ontology combinations.

Related work in the field of Software Customization, RE, Interactive RE and Ontologies is surveyed in Chapter 2. The thesis problem statement, along with the proposed method of Ontology Combination as the solution, is covered in Chapter 3. Chapter 4 covers the details of the implementation and goes over the experiments conducted with the proposed methodology. It also lists the contributions of this thesis. Chapter 5 concludes the thesis and points out some future directions of continuing research.
Chapter 2

Related Work

2.1 Overview

This chapter surveys the various topics pertinent to the thesis and a literature review of related work.

2.2 Software Customization

2.2.1 Overview

This section presents an overview of the research area of Software Customization. It highlights the area of inquiry in the context of background literature.

2.2.2 Software Product Line Engineering

Software Product Line Engineering (SPLE) is defined as a paradigm to develop software applications (software-intensive systems and software products) using platforms and mass customizations [59]. It is divided into two areas: Domain Engineering and Application Engineering. Figure 2.1 summarizes the different processes involved in these two areas.
2.2 Software Customization

Both, Domain and Application Engineering, gather requirements for which some aspects of Requirements Engineering are needed. Domain Engineering is the process of SPLE in which the commonality and the variability of a SPL are defined and realised [59]. It is comprised of five sub-processes: Product Management, Domain Requirements Engineering, Domain Design, Domain Realisation, and Domain Testing. The Domain Requirements Engineering sub-process covers "all activities for eliciting and documenting the common and variable requirements of the product line" [59] whereas the Domain Design sub-process covers activities for defining the reference architecture [59].

Application Engineering is the process of SPLE in which applications of the SPL are built by reusing domain artefacts and exploiting the product line variability [59]. In contrast to Domain Engineering, one of the main goals of Application Engineering is to make use of the commonality and variability of a SPL to develop a customized product line application [59]. Application Engineering is comprised of four sub-processes: Application Requirements Engineering, Application Design, Application Realisation and Application Testing.
2.2 Software Customization

Testing. The Application Requirements Engineering sub-process contains activities that are needed for developing the application requirements specification [59]. Considerable research has been done in the field of SPL in the past few years [51]. Integrating SPL and Software Oriented Architecture (SOA) paradigms has also been an important focal point for researchers, more of which will be covered in a later section. Various open research challenges can be found for topics encompassing SPL [51]. Software factory automation has been proposed [15], analogous to manufacturing factory automation, for managing reusable assets across distinct SPLs. This model is based on an architecture-driven meta-model which is customized to create applications directly. A systematic overview of research literature for product derivation in SPL has also been done [60], where requirements are identified and validated for this purpose.

2.2.3 Service Oriented Architecture

Service Oriented Architecture (SOA) is a software model in which automation logic is decomposed into smaller, distinct units of logic [22]. These units are collectively used to create a larger piece of business automation logic. Figure 2.2 provides an overview of this model.

![Figure 2.2: Service Oriented Architecture](image)

Services can assume different roles when involved in different scenarios [22]. The three main roles are, as shown in Figure 2.2, Service Broker, Service Consumer, and Service Provider. In the role of a Service Provider, a service exposes a public interface
through which it can be invoked by requestors of the service\textsuperscript{[22]}. A Service Consumer is the sender of a service message requesting a specific service\textsuperscript{[22]}. A Service Broker acts as a registry of services, and stores information about what services are available and who may use them. Universal Description, Discovery and Integration (UDDI) is an example of a Service Broker.

The core concept in SOA is that these units can be distributed. They don’t need to reside on the same machine but can be spread across an intranet or even the Internet.

\section*{2.2.4 Integrating SPL and SOA}

SPL and SOA integration is an active area of research. The various studies done in this combined field over the last decade have been surveyed\textsuperscript{[52]}. The studies have been classified according to research focus, types of research and contribution, along with the various fields of ongoing research.

The concepts of SPL, SOA and component frameworks have been compared\textsuperscript{[32]}, concluding with the assertion that while there are differences between them, these concepts are in fact complementary to one another. An approach of a service-oriented architecture in which product lines are regarded as services which are then used to combine together into another, distinct product line has been presented\textsuperscript{[67]}. A web product line to showcase this approach has also been provided there. An approach for reusing and combining services into service oriented product line applications has also been proposed\textsuperscript{[43]}. Various issues such as identification of services are resolved by using feature-oriented product line engineering. Another method has been proposed\textsuperscript{[37]} in which services and their level of granularity are identified by using ontologies in product lines. A way of grouping features and evaluating services, along with a case study, has also been provided there.

Developing SOA applications as SPLs has been attempted\textsuperscript{[49]}. A combination of these two concepts is shown to provide advantages such as improved reuse and production of customized applications for specific clients. The issue of service identification for service-oriented product lines has been explored\textsuperscript{[36]}. An approach has been de-
2.3 Requirements Elicitation

fined which bridges Feature Models (FMs) in SPLs and Business Process Models (BPMs) in SOAs by using a BPM workflow model to identify services.

A model using SOA architecture derived from current software artefacts has been defined [57]. There the focus has been on the reuse of these artefacts as SOA components and the derivation process that assembles products out of services automatically. This proposed approach has been implemented in the form of the Software Product Line Integration Tool (SPLIT) [56], which has been used to develop modular services obtained automatically from existing software artefacts. Then out of these services, products are assembled using a variability-driven derivation process.

2.3 Requirements Elicitation

2.3.1 Overview

This section goes over the relevant research work done in the field of Requirements Elicitation (RE). It also covers the use of Ontologies in RE.

2.3.2 Requirements Elicitation

Requirements Engineering is comprised of activities related to the development and agreement of the final set of Requirements Specifications [68]. The various processes in Requirements Engineering are outlined in Figure 2.3. The main processes used for a majority of projects are: Requirements Elicitation, Requirements Analysis and Requirements Specification. Other processes, such as Requirements Prototyping, are also done for projects where it is feasible to do so. Requirements Elicitation (RE) is defined as the process of discovering the requirements for a system by communicating with customers, system users, and others who have a stake in the development of the system [63]. It requires specific knowledge of the problem along with application domain and organizational knowledge. RE plays an important part in Requirements Engineering.

Traditionally, human communication has been the method of acquiring requirements
However, this mode of collecting requirements is ambiguous and a primary source of errors which leads to flawed and incomplete Requirements Specifications. Recognizing this, attempts have been made to use computer-assisted tools to gather requirements \[46\]. Extending this paradigm, a human-machine dialogue interface using natural language promises to reduce errors in the RE process.

In an early work \[28\], various approaches to obtain requirements were presented using insight gained from social science paradigms. A prototype automated SPL engineering environment has been presented which utilizes a product line repository \[29\]. Multiple-view models of SPLs were then used with a Knowledge Based RE Tool to derive a software product. An approach of interactive RE to build customized software based on a SPL has been presented recently \[74\]. An ontology model comprising of knowledge of common and variable assets has been developed, which is then used to obtain abstract requirements models for specific domains. A case study of an online book shopping system has also been incorporated into that study to illustrate the approach \[74\].
2.3 Requirements Elicitation

2.3.3 Ontologies in RE

Ontologies have been defined as "a formal, explicit specification of a shared conceptualization" [65]. They began to be used in Requirements Engineering in the early 1980s [20]. They were used in a variety of domains such as network management [45] and aerospace [40] [24].

Ontologies have been used for Requirements Analysis [35]. There, the incompleteness and inconsistency in a Requirements Specification was determined by using ontologies. The quality of a specification was measured along with predictions made about requirement changes.

Ontology-based reasoning method for RE has also been introduced [21]. Here, requirements were mapped to functions in domain ontology. Then reasoning was applied to check for errors and other potential requirements. Ontology-driven guidance has been used for RE [24]. Evaluation was done based on a domain ontology and a set of requirements. Further progress has been made in manipulating ontologies by combining them. Combinations make an effective use of knowledge encapsulated in different ontologies [71]. A methodology has been established to perform combinations for RE [71]. Ontology-based RE for software customization, in the context of SPLs, has been performed using an interactive approach [75] [74].

Ontologies have been developed for various Requirements Engineering processes using a university course registration web application system as a case study [62]. There, a model called OntoPersonalURM, which uses a multi-step iterative ontology development process, was created for Requirements Engineers. Ontology-based relation mining has been used for Cloud software requirements [34]. Ontologies have also been used for Requirements Specification verification and validation [17]. Similarly, an ontology of requirements has been used in transforming informal requirements into a formal specification [44].

Table 2.1 summarizes the research covered in this section.
## 2.3 Requirements Elicitation

<table>
<thead>
<tr>
<th>Area</th>
<th>Ontology used for</th>
</tr>
</thead>
</table>
| **Requirements Engineering**| - Use of ontologies in RE began in early 1980s [20]  
- Network management [45]    
- Aerospace [40] [24]        
- University course registration web application [62]  
- Cloud software [34]        |
| **Requirements Elicitation**| - Reasoning method [21]  
- Evaluation [24]  
- Software customization using an interactive approach for Software Product Lines [75] [73] [74]  
- Combining ontologies [71] |
| **Requirements Analysis**   | - Quality of a Requirements Specification and requirement changes [35]  
- Domain knowledge and semantics [53] |
| **Requirements Specification**| - Verification and Validation [17]  
- Transforming informal requirements into a formal Requirements Specification [44] |

Table 2.1: Summary of Ontologies in RE
2.4 Interactive Requirements Elicitation

Recently, significant progress has been made towards interactive RE using ontologies. An interactive machine-guided elicitation of requirements has been developed for the customization of a SPL for SOA based software [74]. An ontology-based requirements model has been developed [73], as shown in Figure 2.4. Three main concepts have been identified in the model - Requirement, Function and Quality. Other concepts have been included as extensions - Softgoal, Rank and OtherInfo. Seven relationships have been developed - Generalize, Decompose, Rely, Contradict, Associate, HasRank, and Invalid. A group of ontology rules has also been developed for RE and ontology instantiation to retrieve implicit knowledge of a product line [74]. A nine-step process has been outlined for instantiating a domain model of a service-oriented architecture of a family of software products.

![Ontology-based Requirement Model](image)

Figure 2.4: Ontology-based Requirement Model [73]

Here, RE is performed using a dialogue-based system. Ontologies are utilized for dialogue management. An ontology model [70] is used to manage dialogue interaction independently of domains [74]. In a related work, similar technique is applied to create customized software using conversational agents based on natural language interaction [73]. The algorithm for the RE process is shown in Figure 2.5 [73]. Complete
2.4 Interactive Requirements Elicitation

details of the evaluation process, along with additional algorithms cited in Figure 2.5 are described in that work [75].

![Pseudo code for requirement evaluation process](image)

Figure 2.5: Pseudo code for requirement evaluation process [73]

After the final set of requirements have been obtained, their service descriptions are converted into an OWL-S ontology. Figure 2.6 shows the overview of the entire system [73].
2.4 Interactive Requirements Elicitation

Figure 2.6: Interactive Requirements Elicitation system [73]
2.5 Ontology and Operations

2.5.1 Overview

This section covers the definition of Ontology as well as the various operations performed on them.

2.5.2 What is an Ontology?

As mentioned in Section 2.3.3, an Ontology is "a formal, explicit specification of a shared conceptualization" [65]. Ontologies enable knowledge sharing and reuse in a specific format. They have the advantage of being a formal and machine manipulable model of a domain of interest. Ontologies present a shared vocabulary in representing domain knowledge which allows reasoning to be performed.

Figure 2.7 shows an example of an ontology as a semantic network. Here, the ontology is modeled as Concepts and Relationships. Concepts can be abstract that represent intentions, beliefs, feelings etc., or they can be specific such as people, computers, tables, etc [30]. Relationships represent a type of association between Concepts of a domain [30]. In Figure 2.7, the Concepts are shown as ovals and the arrows designate the Relationships between the Concepts.

![Figure 2.7: Ontology as a Semantic Network](27)

The same ontology is presented as a UML model in Figure 2.8. The boxes represent the Concepts and the lines between them represent Relationships. An excerpt of the
ontology represented in Web Ontology Language (OWL) format is shown in Figure 2.9. OWL is an ontology language for the Semantic Web with formally defined meaning.

Figure 2.8: Ontology as a UML model

```
<owl:Class rdf:ID="Event"/>
<owl:Class rdf:ID="Album"/>
<owl:Class rdf:ID="Instrument"/>
<owl:Class rdf:ID="Musician"/>
<owl:Class rdf:ID="Admirer"/>
<owl:ObjectProperty rdf:ID="author">
    <owl:inverseOf>
        <owl:ObjectProperty rdf:ID="opus"/>
    </owl:inverseOf>
    <rdfs:domain rdf:resource="#Album"/>
    <rdfs:range rdf:resource="#Musician"/>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:ID="player">
    <rdfs:range rdf:resource="#Musician"/>
    <rdfs:domain rdf:resource="#Instrument"/>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:ID="loudness">
    <rdf:type
    rdf:resource="http://www.w3.org/2002/07/owl#FunctionalProperty"/>
    <rdfs:domain rdf:resource="#Instrument"/>
</owl:ObjectProperty>
...```

Figure 2.9: Ontology represented in OWL (excerpt)
2.5 Ontology and Operations

2.5.3 Operations on Ontologies

Over the previous decades of research, various operations on Ontologies have been identified [23]. They are: Matching, Alignment, Mapping, Integration and Merging.

2.5.3.1 Matching

Matching is the process of finding relationships or correspondences between entities of different ontologies [23]. This area of research is becoming increasingly important for knowledge bases and the Semantic Web. Matching can be performed on Concepts, Attributes, and Relations of ontologies.

Figure 2.10 gives an example of Ontology Matching [42]. Figure 2.10a shows how concepts in the domain of the Motion Picture industry are represented in two different ontologies, $O_1$ and $O_2$. Relations within the two ontologies are also shown as arrows. The dotted lines represent the output of the Matching process. Similarly, in Figure 2.10b, ontologies $O_1$ and $O_2$ contain knowledge of the Food domain. Concepts and relations are matched between them and shown as dotted lines.
2.5 Ontology and Operations

(a) Movie ontologies $O_1$, $O_2$ and matching results. Dotted lines mean a matching

(b) Food ontologies $O_1$, $O_2$ and matching results. Dotted lines mean a matching

Figure 2.10: Ontology Matching [42]
2.5.3.2 **Alignment**

Ontology Alignment is the process of bringing ontologies into agreement through the automatic discovery of mappings between related concepts [31]. It is a set of correspondences between two or more ontologies. The underlying principle in Alignment is that ‘ontologies can approximate other ontologies and that ontologies to be matched are approximation of a common ideal ontology’ [23].

An example of Alignment is given in Figure 2.11. The excerpt shown in this figure is the Alignment of two ontologies: the one on the left side is a fragment of the *Forest Fire Sensor* ontology and the one on the right side is a fragment of the *Fire Trucks Sensor* ontology. The dashed lines denote the Alignment obtained after applying an ontology alignment algorithm [25].

![Figure 2.11: Partial view of an Ontology Alignment](25)
2.5.3.3 Mapping

Mapping is the oriented, or directed, version of the alignment which maps entities of one ontology to at most one entity of another ontology [23]. This can viewed as a collection of mapping rules oriented in a particular direction - from one ontology to another.

Figure 2.12 shows an example of Mapping. Both ontologies, o1 and o2, represent knowledge in the Restaurant domain. Ontology o1 encodes that knowledge in the context of American restaurants, whereas ontology o2 does this in a Japanese context. The bold arrows represent the map generated between the two ontologies. Figure 2.12b presents an abstract view of the Mapping.

Figure 2.12: Ontology Mapping
2.5.3.4 Integration

Integration is the inclusion in one ontology of another ontology [23]. The integrated ontology contains the knowledge of the original ontologies. Integration is different from Merging as one of the ontologies is modified whereas Merging creates a new ontology.

An example of Integration is given in Figure 2.13. $A$ and $B$, are the initial ontologies. Integration results in $B$ being ‘absorbed’ into $A$.

Figure 2.13: Ontology Integration [50]
2.5 Ontology and Operations

2.5.3.5 Merging

Merging is the creation of a new ontology from two, possibly overlapping, source ontologies \[23\]. The initial ontologies are not modified, with the new ontology incorporating the knowledge of both the ontologies.

Figure 2.14 shows an example of Merging. Sample ontology 1 and Sample ontology 2 consist of information about the domain of Cars. A third ontology generated after the Merging, as shown in Figure 2.14c, contains the knowledge of both Sample ontology 1 and Sample ontology 2 as a single ontology.

The ideas behind Ontology Merging can be traced back to the beginning of 1980s \[18\]. The SMART algorithm was an early semi-automatic approach to Ontology Merging and Alignment \[54\]. The PROMPT algorithm was an improvement of SMART and during its development various Ontology Merging operations were identified \[55\].

Mathematical frameworks have been applied to Ontology Merging. Merging has been done using Formal Concept Analysis (FCA-MERGE) \[66\]. Also, Category theory \[33\] has been applied towards merging and Simple PushOut (SPO) in algebraic graph transformation \[48\] has been used to merge ontologies. Description Logic (DL) based merging of Concrete and Fuzzy ontologies has also been accomplished \[41\].

An ontology integration process has been proposed in which two ontologies are merged by generating an ontology intersection containing the maximum number of entities contained in the input ontologies and their corresponding non-contradictory axioms \[69\]. CODE \[26\] is a fully automated system that aims at preserving the source ontology knowledge. It uses natural language processing in combination with a semantic matching approach, along with scenario-based rules to make sure the merging process is accurate. While being very comprehensive, CODE is a holistic process - taking into account all aspects of the source ontologies including Class, Property and Instance. While this is powerful, it is not useful for a more lightweight approach where only the Classes of given ontologies need to be analyzed. It is a quite involved and cumbersome process - going through multiple stages to acquire a merge, and would be difficult to adapt to a nimble setting where quick operations are required.
2.5 Ontology and Operations

(a) Sample Ontology 1 about Car

(b) Sample Ontology 2 about Car

(c) Merged Ontology

Figure 2.14: Ontology Merging [39]
Recently, Cloud-based ontology matching has been provided as a Service for integration and interoperability resolution primarily focused on biomedical systems [16]. A novel approach, but as the system has been built for a distributed architecture of a cloud, it would be difficult to extract and incorporate the technique for a more restrictive environment, such as a traditional, localized desktop system.

Also, ATOM base algorithm has been proposed that takes two ontologies and merges them using an equivalence mapping [61]. A very clear and consistent terminology is presented for the ATOM algorithm and lays down the foundations for developing similar algorithms. Equivalence mapping is clearly defined and applied. A major drawback of the algorithm is that it is limited to an IS-A relationship; it does not take into account other possible relationships.

Table 2.2 presents the various operations and approaches described in the earlier sections. This table compares the various works explored earlier on the basis of whether user intervention is required, the type of relationship that is being used in the work (if explicitly stated) and in the last column of the table, if the work is based on a Design-Time or Run-Time approach.

The following chapter provides further discussion.
<table>
<thead>
<tr>
<th>Work</th>
<th>Automated</th>
<th>Relationship</th>
<th>Place</th>
</tr>
</thead>
<tbody>
<tr>
<td>[Noy and Musen, 1999] (SMART) [54]</td>
<td>✗</td>
<td>-</td>
<td>Design-Time</td>
</tr>
<tr>
<td>[Noy and Musen, 2000] (PROMPT) [55]</td>
<td>✗</td>
<td>-</td>
<td>Design-Time</td>
</tr>
<tr>
<td>[Stumme and Maedche, 2001] (FCA-MERGE) [66]</td>
<td>✗</td>
<td>-</td>
<td>Design-Time</td>
</tr>
<tr>
<td>[Hitzler et al., 2005] (Category theory) [33]</td>
<td>✗</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>[Raunich and Rahm, 2011] (ATOM) [61]</td>
<td>✓</td>
<td>IS-A</td>
<td>Design-Time</td>
</tr>
<tr>
<td>[Fudholi et al., 2014] (CODE) [26]</td>
<td>✓</td>
<td>-</td>
<td>Design-Time</td>
</tr>
<tr>
<td>[Mahfoudh et al., 2014] (Algebraic SPO) [48]</td>
<td>✓</td>
<td>IS-A</td>
<td>Design-Time</td>
</tr>
<tr>
<td>[Wu, 2014] (CODE) [26]</td>
<td>✗</td>
<td>-</td>
<td>Design-Time</td>
</tr>
<tr>
<td>[Amin et al., 2015] (Cloud-based) [16]</td>
<td>✗</td>
<td>-</td>
<td>Design-Time</td>
</tr>
<tr>
<td>[Kumar and Harding, 2015] (Description Logic) [41]</td>
<td>✗</td>
<td>IS-A</td>
<td>Design-Time</td>
</tr>
<tr>
<td></td>
<td></td>
<td>TYPE-OF</td>
<td>PART-OF</td>
</tr>
</tbody>
</table>

Table 2.2: Comparison of Ontology Operations
Chapter 3

A New Method of Ontology Combination

3.1 Overview

This chapter covers the problem statement of this thesis and the proposed methodology as the solution. The specific details of the methodology are developed thoroughly. The corresponding Combine algorithm is covered comprehensively in the later sections.

3.2 Problem Statement

The interactive approach for RE, outlined in Section 2.4 above, uses a single ontology for modeling the domain requirements [73]. This can be further enhanced by acquiring knowledge from other domains. Multiple ontologies can be brought together for this purpose, enabling the approach to acquire additional knowledge from diverse domains.

Various operations on ontologies were covered in Section 2.5.3. Among these approaches, mathematical frameworks [66] [33] [48] require formulating the ontologies into mathematical structures such as lattices and fuzzy structures. This requires an extra ‘overhead’ operation, which is expensive in terms of the additional time that is
required. While bringing mathematical precision to the merging process, the present mathematical approaches lose flexibility and would fall short in performance during a dynamic use of such approaches.

All semi-automatic approaches \[54\] [55] [66] [33] [69] [16] [41] require human intervention at important stages of the merging process. Dynamic combination of ontologies, needed during run-time, should not require any human intervention. Any technique requiring user decisions during the merge process would defeat the purpose of interactive RE as the focus of the user needs to be on gathering requirements rather than merging ontologies. Also, in order to merge ontologies in this manner, the user will need to have an in-depth knowledge of the merging process. For the purpose of a software customization system, this should not be required and this, ideally, should be transparent to the user. The user should not be required to know about ontologies or of the process of merging ontologies; this should be taken care of in the background of the interactive system without involving the user.

Furthermore, all of them are design-time operations. As such, they cannot be applied to an interactive mode of knowledge extraction.

This thesis proposes a dynamic run-time operation of Ontology Combinations which, by overcoming these limitations, can enhance interactive RE immensely.

### 3.3 Ontology Combination

As outlined in Section 2.5, existing approaches are mainly focused on ‘deep’ merges of ontologies. Classes, relations, etc (some or all attributes of ontologies) are sought to be merged. RE does not need this as Requirements Artefacts are usually discrete items brought together to form a new system. An artefact is usually defined as a specification of a physical piece of information that is used or produced by a software development process [64]. RE needs ontology combinations so that new Requirements Specifications can be generated quickly from different ontologies. Reasoning should be relatively quick and ideally should have a minimal overhead in generating a combined ontology as the
entire structure of an ontology does not need to be merged.

Ontology Combination is similar to Ontology Merging but not the same operation. It is different as ontologies being combined together might not share any ideas except for the need of creating a joined ontology that might serve an entirely different purpose from that of the original two ontologies.

3.4 Example

A general example can be used to demonstrate Ontology Combinations. Figure 3.1 shows a Pizza ontology. It has, among other concepts, a Food concept. This concept is further extended to concepts like Pizza, IceCream, PizzaTopping etc. Now, another ontology can also contain information about food items such as the Food ontology shown in Figure 3.2. This ontology contains a concept EdibleThing. EdibleThing is refined into different types of consumable items such as Dessert, SweetFruit etc. If both these ontologies are combined, then a reasoning system based on the Pizza ontology can take advantage of the knowledge available in the Food ontology. A good instance of this would be in creating new and unexpected pizza topping combinations for ordering Pizzas like a topping of SweetFruit on a SpicyPizza. Figure 3.3 shows one possible pair of nodes to achieve this desired combination.
Figure 3.1: Pizza ontology [10]
Figure 3.2: Food ontology [4]
Figure 3.3: Excerpt of the Combined Ontology
3.5 Proposed Methodology

The methodology of performing an Ontology Combination is detailed in this section. Terminology used is presented in Section 3.5.1 Definitions. Then the steps are delineated. The summary is shown in Figure 3.4.

3.5.1 Definitions

Many similar notations for Ontologies exist in research literature. For this discussion, an ontology is defined as a tuple:

\[ O_i = (C_i, R_i, I_i, A_i) \]

where
- \( C_i \) is the set of concepts,
- \( R_i \) is the set of relationships between the concepts,
- \( I_i \) is the set of instances,
- \( A_i \) is the set of axioms.

\( K_{RE} \) is a RE Knowledge Base holding ontologies and their instances specialized towards the acquisition of requirements. \( R_i \) and \( A_i \) are specific to \( O_i \) and \( I_i \) is part of \( K_{RE} \) and therefore will not be considered here.

Instances of ontologies will be assumed to honor the ontological evaluations after the combination. It is assumed that all the ontologies are consistent before the beginning of the combination process. Only concepts, \( C_i \), are needed for the combination and will be analyzed here.

For the sake of brevity and simplicity, a combination of only two ontologies will be delineated here. Starting with primary ontology \( O_p = (C_p, R_p, I_p, A_p) \), a secondary ontology \( O_s = (C_s, R_s, I_s, A_s) \), is selected from \( K_{RE} \). The steps taken to combine them are explained in the following sections.
3.5 Proposed Methodology

3.5.2 Step 1: Generate Correspondences

A set of concept correspondences is defined as the set of (match) mapping between two ontologies [61]. Given two concepts, $p \in C_p$ and $s \in C_s$, a concept correspondence $t$, is defined as an ordered pair $(p, s)$ of a primary ontology concept $p$ and a secondary ontology concept $s$ [61]. Each $t$ is characterized by a type selected from equivalence, is-a and inverse-isa [61]. An equivalence correspondence is defined as a correspondence where $p$ and $s$ represent the same concept; an is-a correspondence is defined as a correspondence where $p$ is a subclass of $s$ and an inverse-isa correspondence is defined as a correspondence where $s$ is a subclass of $p$ [61]. An is-a correspondence is an oriented correspondence from a source concept to a target concept and expresses an is-a relationship between them [61]. An inverse-isa correspondence is similarly defined as the source concept being a ‘superclass’ of the target concept [61]. Here, a set of concept correspondences, $T$, will be used to identify the concepts in primary and secondary ontologies. $T$ will be used to generate the links between them in Step 2.

3.5.3 Step 2: Generate Relationships

On the basis of the type of correspondences, relationships can be generated for the links that tie the ontologies together.

3.5.4 Step 3: Check consistency of combined ontology, $O_c$

The combined ontology, $O_c$, obtained after Step 2, will then be checked for consistency using a suitable reasoner.

3.5.5 Step 4: Validation of $O_c$

A simple reasoning test can be performed to ensure that the link produced is valid and produces sensible results.
Ontology-based requirements elicitation can be then be carried out using the combined ontology $O_c$. After this, if a suitable set of requirements has not yet been obtained, this process can be iterated over again. The above steps can be iterated over as many times as needed until a satisfactory set of requirements is gathered.

As the use of this methodology matures, existing ontologies can be modified and newer ones can be added to $K_{RE}$. Over time, such a methodology would yield a mature collection of ontologies which would help in refining requirements even further, leading to a less ambiguous and a detailed set of Requirements Deliverables.

The next section, covering the Combine algorithm, gives the details of the algorithms involved in this methodology.
3.5 Proposed Methodology

The proposed methodology involves several steps:

1. **Primary Ontology, \( O_p \)**
2. **Create Combined Ontology**
   - **Step 1:** Generate Correspondences
   - **Step 2:** Generate Relationships
   - **Step 3:** Consistency check of combined ontology, \( O_c \)
   - **Step 4:** Validation of \( O_c \)
3. **Perform Requirements Elicitation using \( O_c \)**
4. **Have the Requirements Deliverables been satisfactorily completed?**
   - **Yes** → End
   - **No** → Detach previously combined ontologies, if needed

\( O_c \) is now \( O_d \)

Figure 3.4: Methodology
3.6 Design of Algorithms

3.6.1 Overview

The Combine algorithm is called during the process of RE for a SPL, represented by the ontology, $O_{SPL}$. Another ontology, $O_i$, from the RE Knowledge Base, $K_{RE}$, is given as input to the algorithm to perform the combination.

The algorithm uses the following:

- **WordNet** [13] - is a lexical database for the English language, often described as a combination of a dictionary and thesaurus.

- **Java WordNet Library (JWNL)** [7] - is a free and open-source Java API for accessing WordNet.

- **Apache Lucene** [2] - is a free and open-source information retrieval Java library.

- **SimMetrics** [12] - is a free and open-source Java library of similarity and distance metrics for strings.

The Combine algorithm is composed of smaller algorithms - SelectLink, GetCorrespondences, GetRelationship, FindRelationshipJWNL and GetHighestCM - all of which are described in this section. The SelectLink algorithm is called initially with string, $strSPLLeafNode$ of the leaf node, $V_i$ in $O_{SPL}$ and the ontology to be combined, $O_i$. The algorithm then calls the GetCorrespondences algorithm to get Correspondences, if they exist, between $strLeafNode$ and any node in $O_i$. The GetCorrespondences algorithm in turn, calls the GetRelationship algorithm which tries to find the relationships (IS-A, TYPE-OF, and PART-OF). It does this through the use of the FindRelationshipJWNL algorithm, which uses the JWNL API for WordNet. The GetHighestCM algorithm is used to determine the Correspondence with the highest Confidence Measure in a given set of Correspondences. The SelectLink algorithm returns a Correspondence which is used to link $O_{SPL}$ and $O_i$ together. This enables the two ontologies to be linked together dynamically, resulting in a combined ontology. The combined ontology resides in memory. The algo-
Algorithm can be called as many times as needed to combine other ontologies in $K_{RE}$ with the main $O_{SPL}$ ontology.

### 3.6.2 SelectLink algorithm

The SelectLink algorithm is shown below:

#### Algorithm 1 SelectLink

**Input:** $strSPLLeafNode$ is string for the concept of the leaf node $V_i$ in $O_{SPL}$  
**Input:** $O_i$ is ontology that is to be combined  
**Output:** $c_{Highest}$ is a Correspondence that will link $O_{SPL}$ and $O_i$ together

1: $c \leftarrow \emptyset$  
2: for each node $\in O_i$ do  
3: \hspace{1em} $c \leftarrow c \cup \text{GetCorrespondences}(strSPLLeafNode, node.label)$  
4: end for  
5: $c_{Highest} \leftarrow \text{GetHighestCM}(c)$  
6: return $c_{Highest}$

The input for the SelectLink algorithm is the string for the concept of the leaf node $V_i$ in $O_{SPL}$, $strSPLLeafNode$, and the ontology that is to be combined, $O_i$. It returns as output, a Correspondence $c$, which contains the node in $O_i$ and the relationship that will link $O_{SPL}$ and $O_i$ together.

The loop in Line 2 to Line 4 iterates over all the nodes in $O_i$ to find out if there is a correspondence between the node and $strSPLLeafNode$. This is done by calling the GetCorrespondences algorithm (Section 3.6.3). All correspondences are collected into the set of Correspondences, $c$. The Correspondence with the highest Confidence Measure is selected by calling the GetHighestCM algorithm in Line 5. Confidence Measure is described in GetHighestCM algorithm section (Section 3.6.6). This Correspondence is then returned in Line 6 as the link between $O_{SPL}$ and $O_i$. 


3.6.3 GetCorrespondences algorithm

The GetCorrespondences algorithm is shown below:

Algorithm 2 GetCorrespondences

Input: \( strSPL \) is string for the concept of the leaf node \( V_i \) in \( O_{SPL} \)
Input: \( strI \) is the string for the concept from \( O_i \)
Output: \( c \) is a Correspondence

1: if \( strSPL = strI \) then \( \triangleright \) same string
2: \( c\text{.Relationship} \leftarrow IS-A \)
3: \( c\text{.CM} = 0 \)
4: return \( c \)
5: end if

6: \( tokensSPL \leftarrow \emptyset \)
7: \( tokensI \leftarrow \emptyset \)
8: \( cw \leftarrow \emptyset \)

9: \( strSPL \leftarrow \text{Lucene.StopWordsFilter}(strSPL) \)
10: \( tokensSPL \leftarrow \text{Lucene.Tokenize}(strSPL) \)

11: \( strI \leftarrow \text{Lucene.StopWordsFilter}(strI) \)
12: \( tokensI \leftarrow \text{Lucene.Tokenize}(strI) \)

13: for each \( wSPL \in tokensSPL \) do \( \triangleright \) first pass - try to find Anchor Word
14: \hspace{1em} for each \( wI \in tokensI \) do
15: \hspace{2em} if \( 0.0 \leq \text{SimMetrics.JaroWinkler}(wSPL, wI) \leq 0.1 \) then
16: \hspace{3em} \( w\text{AnchorWord} \leftarrow \text{shortestOf}(wSPL, wI) \)
17: \hspace{2em} if \( \text{sizeof}(tokensSPL) = \text{sizeof}(tokensI) = 1 \) then \( \triangleright \) only one word
18: \hspace{3em} \( c\text{.Relationship} \leftarrow IS-A \)
19: \hspace{2em} return \( c \)
20: \hspace{1em} end if
21: \hspace{1em} end if
22: \hspace{1em} end for
23: \hspace{1em} end for

24: for each \( wSPL \in tokensSPL \) do \( \triangleright \) second pass
25: \hspace{1em} for each \( wI \in tokensI \) do
26: \hspace{2em} \( cw \leftarrow cw \cup \text{GetRelationship}(wSPL, wI) \)
27: \hspace{1em} end for
28: \hspace{1em} end for

29: \( cw\text{Highest} \leftarrow \text{GetHighestCM}(cw) \)

30: if \( \text{cw.size()} \leq 1 \) and \( \text{cwHighest.RelationshipFound} = \text{true} \) then
31: \( c = cw\text{Highest} \)
32: else
3.6 Design of Algorithms

33: if wAnchorWord then
34: if strSPL.length > strI.length then \(\triangleright\) more ‘generic’ concept
35: c.Relationship ← IS-A
36: c.CM = 0
37: else if word before wAnchorWord in strSPL = word before wAnchorWord in strI = VERB) then
38: c.Relationship ← PART-OF
39: c.CM = 0
40: end if
41: else \(\triangleright\) No relationship has been found
42: c.Relationship = NULL
43: end if
44: end if
45: return c

The GetCorrespondences algorithm tries to determine the correspondence between two concepts, strSPL from OSPL and strI from Oi, through two levels of matches:

- **String Level Match** is done on the entire two strings to figure out the relationship between strSPL and strI. Line 1 to Line 5 and Line 13 to Line 23 are the String Level Matching parts in the algorithm.

- **Word Level Match** is done on individual words of the strings to figure the relationship between them. Line 24 to Line 28 are the Word Level Matching parts in the algorithm.

The input to the GetCorrespondences algorithm are strSPL which is the string for the concept of the leaf node Vl in OSPL and strI which is the string for the concept from Oi. The output of the algorithm is a Correspondence which contains the relationship between the two input strings and the Confidence Measure for that relationship.

A check is done in Line 1 to see if the two strings are equal and if they are, then the relationship is considered of type IS-A and the algorithm returns this as a Correspondence with Confidence Measure of 0. From Line 9 to Line 12, the Apache Lucene library is used to filter for stop words and to tokenize the input strings. The first pass through the two strings, from Line 13 to Line 23, tries to find an AnchorWord. An AnchorWord is used to determine a ‘core’ concept between the two strings. This is determined by
utilizing the Jaro-Winkler distance implemented in the SimMetrics library. If the words are similar and there is only one word in both the strings, then the relationship is of type IS-A and the correspondence has been found.

If the relationship has not been found yet, then a second pass is made through the string tokens, from Line 24 to Line 28. Each word in tokensSPL is compared with each word in tokensI to determine the relationship between them by calling the GetRelationship algorithm. All the Correspondences obtained are put into the set of Correspondences, cw. The Correspondence with the highest Confidence Measure is selected by calling the GetHighestCM algorithm in Line 29.

If no relationship has yet been found, then a relationship is sought based on the wAnchorWord using the logic between Line 32 to Line 44. If strSPL is longer than strI, then strI represents a more ‘general’ concept and thus the relationship is of type IS-A. On the other hand, if the words before the wAnchorWord - both in strSPL and strI are ‘verbs’ (for example, ‘Select book name’ and ‘Find book author’ - here ‘Select’ and ‘Find’ are verbs), then the relationship is assumed to be of type PART-OF.

If no relationship has been found yet and there was no wAnchorWord, then no relationship has been found and the algorithm returns a NULL relationship.
3.6 Design of Algorithms

3.6.4 GetRelationship algorithm

The GetRelationship algorithm is shown below:

**Algorithm 3 GetRelationship**

**Input:** word1 and word2 are strings
**Output:** cw is a correspondence between the two words

1: word1POS ← ∅
2: word2POS ← ∅
3: cwAll ← ∅

4: JWNLI.Initialize()

5: word1POS ← JWNLI.GetPOS() ▷ get all Parts-of-Speech for word1
6: word2POS ← JWNLI.GetPOS() ▷ get all Parts-of-Speech for word2

7: for each p1 ∈ word1POS do
8:   for each p2 ∈ word2POS do
9:     if p1 = p2 then
10:        r ← JWNLI.FindRelationshipSYNONYM()
11:        if r ≠ NULL then
12:           cw.Religation ← IS-A
13:           cw.ConfidenceMeasure ← r.Depth
14:           cwAll ← cwAll ∪ cw
15:      end if
16:      r ← JWNLI.FindRelationshipHYPERNYM()
17:      if r ≠ NULL then
18:         cw.Religation ← TYPE-OF
19:         cw.ConfidenceMeasure ← r.Depth
20:         cwAll ← cwAll ∪ cw
21:      end if
22:     if p1 = p2 = VERB then
23:        r ← JWNLI.FindRelationshipTROPONYM()
24:        if r ≠ NULL then
25:           cw.Religation ← TYPE-OF
26:           cw.ConfidenceMeasure ← r.Depth
27:           cwAll ← cwAll ∪ cw
28:      end if
29:     endif
30:   end if
31: end if

32: if p1 = p2 then
33:   r ← JWNLI.FindRelationshipTROPONYM()
34:   if r ≠ NULL then
35:      cw.Religation ← TYPE-OF
36:      cw.ConfidenceMeasure ← r.Depth
37:      cwAll ← cwAll ∪ cw
38:   end if
39: end if
3.6 Design of Algorithms

35: \textbf{endif}
36: \textbf{r} \leftarrow \text{FindRelationshipJWNL(HOLONYM)}
37: \textbf{if } \textbf{r} \neq \text{NULL} \textbf{ then}
38: \textbf{cw}.\text{Relationship} \leftarrow \text{PART-OF}
39: \textbf{cw}.\text{ConfidenceMeasure} \leftarrow \textbf{r}.\text{Depth}
40: \textbf{cwAll} \leftarrow \textbf{cwAll} \cup \textbf{cw}
41: \textbf{endif}
42: \textbf{r} \leftarrow \text{FindRelationshipJWNL(MERONYM)}
43: \textbf{if } \textbf{r} \neq \text{NULL} \textbf{ then}
44: \textbf{cw}.\text{Relationship} \leftarrow \text{PART-OF}
45: \textbf{cw}.\text{ConfidenceMeasure} \leftarrow \textbf{r}.\text{Depth}
46: \textbf{cwAll} \leftarrow \textbf{cwAll} \cup \textbf{cw}
47: \textbf{endif}
48: \textbf{endif}
49: \textbf{endfor}
50: \textbf{endfor}
51: \textbf{cwHighest} \leftarrow \text{GetHighestCM(\textbf{cwAll})}
52: \textbf{return } \textbf{cwHighest}

The GetRelationship algorithm takes as input two strings, \textit{word1} and \textit{word2} and returns as output a \textit{Correspondence Word}, \textit{cw} which contains the relationship between the two words and the corresponding \textit{Confidence Measure}.

The JWNL library is initialized in Line 4. In Line 5 and Line 6, all the possible Parts-Of-Speech values for the given words are found, as it is possible for a word to be a NOUN or a VERB depending on the context. For example, the word ‘act’ can be used as a NOUN as in “Act II of Hamlet” and as a VERB - “acting in a movie”. In the following part of the algorithm, from Line 7 to Line 50, all the Parts-Of-Speech found for each of the word is iterated through and if a match is found for the Part-Of-Speech, then the relationship and the \textit{Confidence Measure} for that relationship is determined by calling the FindRelationshipJWNL algorithm with different WordNet Pointer Types such as SYNONYM, MERONYM, etc.

Since there can be multiple Parts-Of-Speech for the two words, the \textit{Correspondence} with the highest \textit{Confidence Measure} is obtained by calling the GetHighestCM algorithm in Line 51, which is then subsequently returned by the algorithm.
The following table summarizes the various WordNet Pointer and Relationship types:

<table>
<thead>
<tr>
<th>WordNet Pointer Type</th>
<th>Relationship Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>SYNONYM (SIMILAR)</td>
<td>is-a</td>
</tr>
<tr>
<td>HYPONYM, HYPERNYM,</td>
<td>type-of</td>
</tr>
<tr>
<td>TROPONYM</td>
<td></td>
</tr>
<tr>
<td>HOLONYM, MERONYM</td>
<td>part-of</td>
</tr>
</tbody>
</table>

Table 3.1: WordNet and Relationships
3.6 Design of Algorithms

3.6.5 FindRelationshipJWNL algorithm

The FindRelationshipJWNL algorithm is shown below:

Algorithm 4 FindRelationshipJWNL

Input: start and end are JWNL.IndexWords
Input: type is the type of relationship being inquired about
Output: r is a JWNL.Relationship

1: JWNL.Synset[] startSenses = start.getSenses()
2: JWNL.Synset[] endSenses = end.getSenses()
3: JWNL.Relationship r ← NULL
4: for each s1 ∈ startSenses do
5:     for each s2 ∈ endSenses do
6:         RelationshipList = JWNL.RelationshipFinder(startSenses[i], endSenses[j], type)
7:         if RelationshipList ≠ NULL then
8:             r ← RelationshipList[0]
9:         return r
10:     end if
11: end for
12: end for
13: return NULL

The FindRelationshipJWNL algorithm takes as input two JWNL.IndexWords, start and end. It also needs the type of relationship that needs to be figured out (as shown in the table in Section 3.6.4), designated as type. If the requested relationship is found, the algorithm returns as output a JWNL.Relationship, r which contains the relationship between the two words and the corresponding Confidence Measure. Otherwise, it returns a NULL value.

The WordNet senses for the two input words are retrieved in Line 1 and Line 2. Between Line 4 and Line 12, the senses for each word are iterated over to find a match for type. If a match is found, then the algorithm returns that relationship. Otherwise, a NULL value is returned.
3.6.6 GetHighestCM algorithm

The GetHighestCM algorithm is shown below:

Algorithm 5 GetHighestCM

Input: \( c \) is set of Correspondences
Output: \( c_{\text{Highest}} \) is a Correspondence that has the highest Confidence Measure

1: \( c_{\text{Highest}}.\text{RelationshipFound} \leftarrow \text{false} \)
2: \( \text{if } c.\text{size()} \neq 0 \text{ then} \)
3: \( c_{\text{Highest}} \leftarrow c.\text{get}(0) \)
4: \( \text{for each } c_1 \in c \text{ do} \)
5: \( \text{if } c_1.\text{RelationshipFound} = \text{true} \text{ then} \)
6: \( \text{if } c_{\text{Highest}}.\text{ConfidenceMeasure} \geq c_1.\text{ConfidenceMeasure} \text{ then} \)
7: \( c_{\text{Highest}} \leftarrow c_1 \)
8: \( \text{end if} \)
9: \( \text{end if} \)
10: \( \text{end for} \)
11: \( \text{end if} \)

12: \( \text{if } (c.\text{size()} > 1) \text{ and } (c_{\text{Highest}}.\text{CM} > \text{CM_THRESHOLD}) \text{ then} \)
13: \( c_{\text{Highest}}.\text{Relationship} \leftarrow \text{NULL} \)
14: \( \text{end if} \)
15: \( \text{return } c_{\text{Highest}} \)

If more than one word match/relationship has been found (Line 2), then the Confidence Measure (CM) is used to resolve the relationship in the loop from Line 4 to Line 10. This measure represents the depth between the two words/concepts and comes directly from the JWNL library. The closer the Confidence Measure is to 0, the closer it is assumed to the ‘real’ relationship. A Confidence Measure of 0 represents a direct match. If two (or more) relationships have the same Confidence Measure, the last one is selected as the relationship between the two words. Similarly, if there are multiple word correspondences between two strings, the Confidence Measure is used to determine the eventual relationship between the two strings.

In Line 12 if the Confidence Measure found is greater then the CM_THRESHOLD, then it is determined that no relationship has been found. WordNet provides senses between two words that can very deep, and as such can provide very obscure relationships, which perhaps makes sense at some literary level but may not be useful in the normal
usage of the language. $CM\_THRESHOLD$ is thus used to ensure that no arcane or vague relationships are provided as output by the algorithm.

### 3.6.7 Time Complexity

**Note:** The discussion below follows the convention where the symbol $V$ is used as a shorthand to denote $|V|$ (the number of vertices in a graph) in the context of asymptotic notation for Graph Algorithms [19].

For the Combine algorithm,

- $V_i$ is the set of nodes in the ontology $O_i$,
- $m$ is the number of words in $tokens_{SPL}$,
- $n$ is the number of words in $tokens_I$

The Combine algorithm is composed of the following algorithms: SelectLink, GetCorrespondences, GetRelationship, FindRelationshipJWNL and GetHighestCM algorithms. Since WordNet is a finite set of words and their senses, interaction with the WordNet database in GetRelationship and FindRelationshipJWNL algorithms is assumed to be of constant time for the purpose of determining the time complexity of the Combine algorithm.

The GetHighestCM algorithm contains the for loop in Line 4 which iterates over a given set of all the Correspondences to find out the highest Confidence Measure. This algorithm is primarily called from two places - in the SelectLink algorithm to determine the Correspondence to link $O_{SPL}$ and $O_i$ together and in the GetCorrespondences algorithm to obtain the correspondences between $str_{SPL}$ and $str_I$. When GetHighestCM is called in the SelectLink algorithm, the set of Correspondences can hold, at maximum, a Correspondence for each node in $O_i$, which is $V_i$. When the algorithm is called from the GetCorrespondences algorithm, the set of Correspondences can hold, at maximum, $mn$ Correspondences.

SelectLink and GetCorrespondences are the main algorithms which have a direct im-
pact on the time complexity of the whole Combine algorithm. The SelectLink algorithm goes through all the nodes in $O_i$ to determine if any Correspondence exists between the leaf node in $O_{SPL}$ and $O_i$. Therefore, it will always loop for $V_i$ iterations. The critical part of the Combine algorithm is the GetCorrespondences algorithm. It is where the decision is made for the relationship between the two nodes using the two strings, one from $O_{SPL}$ and the other from $O_i$. The GetCorrespondences algorithm tries find a Correspondence between $str_{SPL}$ and $str_I$. The loops - between Line 13 and Line 23, Line 24 and Line 28 - go over the tokens generated for $str_{SPL}$ and $str_I$, $m$ and $n$ times. For GetCorrespondences algorithm, $(mn + mn) = 2mn$ which in turn, implies $mn$ operations.

From the above analysis, it can be seen that for the average case, the time complexity of the Combine algorithm is $O(mnV_i)$. In the worst case, the length of both $str_{PL}$ and $str_I$ can be the same, yielding a value of $O(n^2V_i)$. In the best case scenario, $str_{SPL}$ or $str_I$ or both, can have just one word, which would then yield a time-complexity of $O(kV_i)$, assuming $k = mn$. The various cases are summarized below:

<table>
<thead>
<tr>
<th>Case</th>
<th>Time Complexity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Average Case</td>
<td>$O(mnV_i)$</td>
</tr>
<tr>
<td>Worst Case</td>
<td>$O(n^2V_i)$</td>
</tr>
<tr>
<td>Best Case</td>
<td>$O(kV_i)$</td>
</tr>
</tbody>
</table>

$k = mn$, where $m = 1$ or $n = 1$

Table 3.2: Time Complexity of Combine algorithm
Chapter 4

Experiments

4.1 Overview

This chapter covers the details of the implementation of the proposed approach. It also covers the details of the experiments undertaken after the implementation. The last section lists the contributions of this study.

4.2 Software

The following is a comprehensive list of all the software, libraries and APIs used for implementing the solution:

- Java [6] - the programming language
- OWL API [8] - Java API for creating, manipulating and serializing OWL ontologies
- Pellet [9] - OWL DL reasoner
4.3 Interface

- **Java WordNet Library (JWNL)** [7] - Java API for accessing WordNet
- **Apache Lucene** [2] - Java library used for information retrieval
- **SimMetrics** [12] - Java library for similarity and distance metrics for strings
- **Apache Ant** [1] - Java library for automating software build processes

The Combine algorithm, as discussed in Chapter 3, was implemented within the existing interactive RE system [75] [73].

4.3 Interface

Figure 4.1 shows the existing interface of the dialogue system [73].

![Figure 4.1: Existing interface of Interactive Requirements Elicitation system](image)

The ‘Ontology Combination Viewer’ (OC Viewer), shown in Figure 4.2, was added to the interface to display the status of the various ontological combinations happening in the background during the session. The output is color-coded to help convey the information quickly:

- **Green** - means the ontology combination was successful
- **Red** - means the ontology combination was unsuccessful
- **Blue** - means that a leaf node was encountered
4.3 Interface

- **Magenta** - means that a mobile platform ontology was successfully combined

![Ontology Combination Viewer (OC Viewer)](image)

Figure 4.2: Ontology Combination Viewer (OC Viewer)

Figure 4.3 shows the initial state of the system. The ‘Ontology Combination Viewer’ (OC Viewer) is shown on the bottom right side. The OC Viewer displays the output of the Combine algorithm during the entire session.

![Initial state of Interactive Requirements Elicitation system](image)

Figure 4.3: Initial state of Interactive Requirements Elicitation system

Figure 4.4 shows the state of the system after some requirements have been selected, with the OC Viewer displaying the corresponding messages on the status of ontology combinations.

49
Figure 4.4: State of Interactive Requirements Elicitation system after selecting some requirements
4.4 Experiments

4.4.1 Scenario I - Single Ontology

As mentioned previously, the existing system [73] works with a single ontology. The Ontology Combination methodology seeks to improve this by letting the system harness other ontologies dynamically at run-time.

4.4.2 Scenario II - Multiple Ontologies

To illustrate the methodology, a scenario of requirements elicitation performed for creating an online bookstore system can be used. BookStore ontology, shown in Figure 4.5, can be assumed as the initial primary ontology $O_{SPL}$ for the new online bookstore system [75]. This ontology represents the knowledge of an online bookstore system. The RE Knowledge Base, $K_{RE}$, contains three more ontologies, Search, OrderSummary and ManagePaymentInfo [75], shown in Figure 4.6, Figure 4.7 and Figure 4.8 respectively. Search, OrderSummary and ManagePaymentInfo are designated as $O_i$, $O_j$ and $O_k$ respectively.
Figure 4.5: BookStore ontology, $O_{SPL}$ in $K_{RE}$
4.4 Experiments

Figure 4.6: Search ontology, $O_i$ in $K_{RE}$

Figure 4.7: OrderSummary ontology, $O_j$ in $K_{RE}$
4.4 Experiments

Figure 4.8: *ManagePaymentInfo* ontology, $O_k$ in $K_{RE}$
4.4 Experiments

The process begins with the primary ontology $O_{SPL}$ being selected in the interactive RE system [73]. As the various requirements are selected, if they are a leaf node, i.e., they don’t rely on or decompose into, other requirements, the Combine algorithm is called for that leaf node. For the SearchBooks leaf node of the $O_{SPL}$, $O_i$ is selected by the Combine algorithm from $K_{RE}$. $O_i$ is then combined with $O_{SPL}$ with an IS-A relationship with the node Search in $O_i$ to form a combined ontology. An excerpt of $O_{SPL}$ after combination is shown in Figure 4.9.

![Figure 4.9: Excerpt of Combined Ontology - $O_{SPL}$ and $O_i$](image)

Figure 4.9: Excerpt of Combined Ontology - $O_{SPL}$ and $O_i$

Figure 4.10 displays the output in the OC Viewer after the $O_{SPL}$ and $O_i$ have been successfully combined. When the "Search Relevant Books" (SearchBooks) leaf node is encountered, the OC Viewer displays the message in Blue. The successful combination of $O_{SPL}$ and $O_i$ is logged in Green and gives the name of the leaf node in $O_{SPL}$ - "Search relevant books" (SearchBooks), the name of the node that was matched in $O_i$ - "Search" (Search), the type of relationship found - "ISA", Confidence Measure of 0.0 and the name (along with the path) of the ontology file - "./KnowledgeBase/Ontology/Search.owl".

![Figure 4.10: OC Viewer output after $O_{SPL}$ and $O_i$ combination](image)
Ontology-based requirements elicitation is then continued on using the combined ontology. As elicitation moves on, once the FinishOrder leaf node is selected, the Combine algorithm is called again. This time $O_j$ is selected from $K_{RE}$. It is combined with $O_{SPL}$ with OrderSummary of $O_j$ using the TYPE-OF relationship. An excerpt of the combined ontology is shown in Figure 4.11.

Figure 4.11: Excerpt of Combined Ontology - $O_{SPL}$ and $O_j$

Figure 4.12 displays the output in the OC Viewer after the $O_{SPL}$ and $O_j$ have been successfully combined. When the “Finish the Order“ (FinishOrder) leaf node is encountered, the OC Viewer displays the message in Blue. The successful combination of $O_{SPL}$ and $O_j$ is logged in Green and gives the name of the leaf node in $O_{SPL}$ - “Finish the Order“ (FinishOrder), the name of the node that was matched in $O_j$ - “Get the Order Summary“ (OrderSummary), the type of relationship found - ”TYPEOF“, Confidence Measure of 0.0 and the name (along with the path) of the ontology file - “./Knowledge-Base/Ontology/OrderSummary.owl“.

Figure 4.12: OC Viewer output after $O_{SPL}$ and $O_j$ combination
After the leaf node of *ChoosePaymentOption* is selected, the Combine algorithm is called again, and this time the *ManagePaymentInfo* ontology, *Ok*, is selected for combination with *OSPL* to help refine the *ChoosePaymentOption* concept. The node *ManagePaymentInfo* in *Ok* is linked to *ChoosePaymentOption* in *OSPL* with a PART-OF relationship. An excerpt of the combined ontology is shown in Figure 4.13. The complete ontology, after the three iterations, is given in Figure 4.15.

![Excerpt of Combined Ontology - OSPL and Ok](image)

**Figure 4.13: Excerpt of Combined Ontology - OSPL and Ok**

Figure 4.14 displays the output in the OC Viewer after the *OSPL* and *Ok* have been successfully combined. When the "Choose Payment Option" (*ChoosePaymentOption*) leaf node is encountered, the OC Viewer displays the message in **Blue**. The successful combination of *OSPL* and *Ok* is logged in **Green** and gives the name of the leaf node in *OSPL* - "Choose Payment Option" (*ChoosePaymentOption*), the name of the node that was matched in *Ok* - "Manage Payment Information" (*ManagePaymentInfo*), the type of relationship found - "PARTOF", **Confidence Measure** of -1.0 and the name (along with the path) of the ontology file - "/KnowledgeBase/Ontology/ManagePaymentInfo.owl".

![OC Viewer output after OSPL and Ok combination](image)

**Figure 4.14: OC Viewer output after OSPL and Ok combination**

After ontology combinations, some relationships might need to be updated. The fol-
Figure 4.15: Complete Combined Ontology after three iterations
4.4 Experiments

Following section discusses this point in the context of combining ontologies for generating specifications for different platforms.
4.4.3 Scenario III - Ontology of Mobile SOA Functions

By including an ontology with platform-dependent functions, Ontology Combination enables the system to be extended to produce specifications for different operating systems. To illustrate this, the PlatformMobile ontology, $O_{\text{Mobile}}$, in Figure 4.16 is used. The $O_{\text{Mobile}}$ ontology has been developed in particular to show how ontology combinations can allow for the dynamic inclusion of concepts that are platform-specific while gathering requirements interactively. It contains knowledge of platform-dependent concepts, such as GetGPSCoordinates and GetMobileWallet, which are specific to mobile (and similar) devices. When the $O_{\text{Mobile}}$ ontology is combined during interactive RE, it allows for the dynamic selection of these platform-dependent requirements.

After a leaf node has been reached, then a search is carried out in $K_{RE}$ for ontologies matching the node with the wildcard character (eg. *locate*). If a corresponding match for the filename is found, then the first matched file is selected for trying the combination (this is the default Ontology Combination approach). If no file has been found, then a second attempt is made with the mobile ontology to find a correspondence between the leaf node and any concepts inside the mobile ontology. This two-step search process ensures that the mobile ontology is tried at least once for combination. With $O_{\text{SPL}}$, the leaf node of ChooseDeliveryOption is matched with DeliveryInfo in $O_{\text{Mobile}}$ using an IS-A relationship. An excerpt of the combined ontology is shown in Figure 4.17.

Figure 4.18 displays the output in the OC Viewer after the $O_{\text{SPL}}$ and $O_{\text{Mobile}}$ have been successfully combined. When the "Choose Delivery Option" (ChooseDeliveryOption) leaf node is encountered, the OC Viewer displays the message in Blue. The successful combination of $O_{\text{SPL}}$ and $O_{\text{Mobile}}$ is logged in Magenta and gives the name of the leaf node in $O_{\text{SPL}}$ - "Choose Delivery Option" (ChooseDeliveryOption), the name of the node that was matched in $O_{\text{Mobile}}$ - "Delivery Information" (DeliveryInfo), the type of relationship found - "ISA", Confidence Measure of 0.0 and the name (along with the path) of the ontology file - "/KnowledgeBase/OntologyMobile/PlatformMobile.owl".
4.4 Experiments

Figure 4.16: *PlatformMobile* ontology, $O_{Mobile}$ in $K_{RE}$
4.4 Experiments

Figure 4.17: Excerpt of Combined Ontology - $O_{SPL}$ and $O_{Mobile}$

Figure 4.18: OC Viewer output after $O_{SPL}$ and $O_{Mobile}$ combination
After the ontologies are combined if the same requirement exists in two ontologies, they are ‘merged’ together. Only one copy of a requirement is inside the combined ontology - the ‘oldest’ one. The ‘incoming’ requirement is ‘merged’ with the old one - the various properties (relyOn, decomposeInto, etc) are updated with the IRIs of the ‘incoming’ requirement. The ‘neighboring’ requirements in the incoming ontology will automatically be linked to the ‘old’ requirement in the combined ontology. The requirement will then be evaluated just like any other regular requirement.

4.5 Case Study

This section provides a case study of an online book shopping software system. The book shopping system is customized through an interactive RE session. Ontology combinations occur during the interactive session, which are shown in the OC Viewer. Finally, an OWL-S file is generated as the output of the customization.

To customize the SOA based online book shopping system, RE is performed using the BookStore ontology, OSpL [75]. The user needs the software system to retrieve information about books [75] and to manage shopping carts as well. So the software system should be able to retrieve details of books and provide shopping functionality such as payment and delivery information. To accomplish this, the interactive RE system is used to perform the customization. Listings 4.1 to 4.11 show the dialogue utterances between the user (U) and the system (M) during the interactive session.

In Listing 4.1 the user is informed that the essential requirement “Manage a shopping cart” has been selected. Similarly, the system informs the user that another essential requirement “Locate a book” has been picked. Following these selections, groups of functional requirements, which are part of a particular requirement, are pre-evaluated. Then, the system prompts the user for choosing various requirements. While selecting requirements, if a leaf node is encountered, then the system automatically combines an ontology from the RE Knowledge Base, KRE, using the Combine algorithm. This is done for the following leaf nodes: “Search relevant books” in Listing 4.2 "Finish the Order"
This is done in a manner that is transparent to the user. After ontology combinations, the new requirements are loaded into the session automatically. The user is offered to choose the new requirement just like any other regular requirement - the user does not know the difference between a requirement from the original ontology \( O_{SPL} \) and a requirement obtained from an ontology in \( K_{RE} (O_i, O_j, O_k, O_{Mobile}) \) which has been combined with the original ontology. The OC Viewer displays the information about ontology combinations in the background during the interactive session. After completion, the user is given the choice of changing his decisions, and if no further changes are needed, then the RE session is finished.
4.5 Case Study

M: Do you like to customize online book shopping service? [YES]

U: YES

M: The functional requirement "Manage a shopping cart" is essential to the system. It will be picked. [OK]

U: OK

M: The functional requirement "Manage a shopping cart" has been picked. [OK]

U: OK

M: The functional requirement "Locate a book" is essential to the system. It will be picked. [OK]

U: OK

M: The functional requirement "Locate a book" has been picked. [OK]

U: OK

M: The functional requirement "Locate a book" is going to be picked. [OK]

U: OK

M: A group of functional requirements, which compose this requirement, are going to be pre-evaluated. [OK]

U: OK

M: The requirement "Get detailed info of a book" need to be evaluated. Would you like to choose the functional requirement "Get detailed info of a book"? [YES, NO]

U: YES

M: Are you sure you want to pick this requirement "Get detailed info of a book"? [YES]

U: YES

M: The group of functional requirements, which compose the requirement "Get a list of relevant books", have been pre-evaluated. [OK]

U: OK

M: The functional requirement "Get a list of relevant books" has been picked. [OK]

U: OK

M: The functional requirement "Search relevant books" has been picked. [OK]

U: OK

M: The functional requirement "Sort books in a list" has been picked. [OK]

U: OK

M: The requirement "Get a list of relevant books" need to be evaluated. Would you like to choose the functional requirement "Get a list of relevant books"? [YES, NO]

U: YES

M: Are you sure you want to pick this requirement "Get a list of relevant books"? [YES]

U: YES

M: The functional requirement "Get publication info" need to be evaluated. Would you like to choose the functional requirement "Get publication info"? [YES, NO]

U: YES

M: Are you sure you want to pick this requirement "Get publication info"? [YES]

U: YES

M: The requirement "Get contents" need to be evaluated. Would you like to choose the functional requirement "Get contents"? [YES, NO]

U: YES

M: The functional requirement "Get a list of relevant books" is going to be picked. [OK]

U: OK

Listing 4.1: Dialogue Utterances (Part 1)

Listing 4.2: Dialogue Utterances (Part 2)
51 M: Are you sure you want to pick this requirement "Get contents"? [YES]
52 U: YES
53 M: The group of functional requirements, which compose the requirement "Get detailed info of a book", have been pre-evaluated. [OK]
54 U: OK
55 M: The functional requirement "Get detailed info of a book" has been picked. [OK]
56 U: OK
57 M: The functional requirement "Get publication info" has been picked. [OK]
58 U: OK
59 M: The functional requirement "Get contents" has been picked. [OK]
60 U: OK
61 M: The functional requirement "Get reference to an item in the shopping cart" has been picked. [OK]
62 U: OK
63 M: The functional requirement "Remove a book from cart" has been picked. [OK]
64 U: OK
65 M: The functional requirement "Add a book to the shopping cart" has been picked. [OK]
66 U: OK
67 M: The functional requirement "Pick an item in shopping list" has been picked. [OK]
68 U: OK
69 M: The functional requirement "Get a shopping list" has been picked. [OK]
70 U: OK
71 M: The functional requirement "Finish the Order" has been picked. [OK]
72 U: OK
73 M: The requirement "Search in book keywords" need to be evaluated. Would you like to choose the functional requirement "Search in book keywords"? [YES, NO]
74 U: YES
75 M: Are you sure you want to pick this requirement "Search in book keywords"? [YES]
76 U: YES
77 M: The functional requirement "Search in book keywords" is going to be picked. [OK]
78 U: OK
79 M: A group of quality requirements, which are related with this requirement, are going to be pre-evaluated. [OK]
80 U: OK
81 M: The requirement "Exact match" need to be evaluated. Would you like to choose the quality requirement "Exact match"? [YES, NO]
82 U: YES
83 M: If you choose to pick the quality requirement "Exact match", 1 requirement, which hasn't been evaluated, is contradicted with this requirement. It cannot be picked. Are you sure you want to pick this requirement "Exact match"? [YES]
84 U: YES
85 M: A group of quality requirements which are related with requirement "Search in book keywords", have been pre-evaluated. [OK]
86 U: OK
87 M: The functional requirement "Search in book keywords" has been picked. [OK]
88 U: OK
89 M: The requirement "Search" need to be evaluated. Would you like to choose the functional requirement "Search"? [YES, NO]
90 U: YES
91 M: Are you sure you want to pick this requirement "Search"? [YES]
92 U: YES
93 M: The functional requirement "Search" is going to be picked. [OK]
94 U: OK
95 M: A group of functional requirements, which compose this requirement, are going to be pre-evaluated. [OK]
96 U: OK
97 M: The requirement "Advanced search" need to be evaluated. Would you like to choose the functional requirement "Advanced search"? [YES, NO]
98 U: YES

Listing 4.3: Dialogue Utterances (Part 3)

Listing 4.4: Dialogue Utterances (Part 4)
4.5 Case Study

M: Are you sure you want to pick this requirement "Advanced search"? [YES]
U: YES

M: The group of functional requirements, which compose the requirement "Search", have been pre-evaluated. [OK]
U: OK

M: The functional requirement "Search" has been picked. [OK]
U: OK

M: The functional requirement "Advanced search" has been picked. [OK]
U: OK

M: The quality requirement "Exact match" has been picked. [OK]
U: OK

M: The quality requirement "Broad match" has been abandoned. [OK]
U: OK

M: The requirement "Get the Order Summary" need to be evaluated. Would you like to choose the functional requirement "Get the Order Summary"? [YES, NO]
U: YES

M: If you choose to pick the functional requirement "Get the Order Summary", 4 requirements, which haven't been evaluated, are prerequisite of this requirement. They will be picked. Are you sure you want to pick this requirement "Get the Order Summary"? [YES]
U: YES

M: The functional requirement "Get the Order Summary" has been picked. [OK]
U: OK

M: The functional requirement "Calculate Total Price" is going to be picked. [OK]
U: OK

M: A group of functional requirements, which compose this requirement, are going to be pre-evaluated. [OK]
U: OK

M: The requirement "Calculate Total Price" need to be evaluated. Would you like to choose the functional requirement "Enter Delivery Information"? [YES, NO]
U: YES

M: Are you sure you want to pick this requirement "Enter Delivery Information"? [YES]
U: YES

M: The requirement "Choose Delivery Option" need to be evaluated. Would you like to choose the functional requirement "Choose Delivery Option"? [YES, NO]
U: YES

M: Are you sure you want to pick this requirement "Choose Delivery Option"? [YES]
U: YES

Listing 4.5: Dialogue Utterances (Part 5)

Listing 4.6: Dialogue Utterances (Part 6)
4.5 Case Study

Listing 4.7: Dialogue Utterances (Part 7)

145 M: The group of functional requirements, which compose the requirement "Set Delivery Information", have been pre-evaluated. [OK]
146 U: OK
147 M: The functional requirement "Set Delivery Information" has been picked. [OK]
148 U: OK
149 M: The functional requirement "Choose Delivery Option" has been picked. [OK]
150 U: OK
151 M: The functional requirement "Enter Delivery Information" has been picked. [OK]
152 U: OK
153 M: The functional requirement "Set Payment Information" is going to be picked. [OK]
154 U: OK
155 M: A group of functional requirements, which compose this requirement, are going to be pre-evaluated. [OK]
156 U: OK
157 M: The requirement "Choose Payment Option" need to be evaluated. Would you like to choose the functional requirement "Choose Payment Option"? [YES, NO]
158 U: YES
159 M: Are you sure you want to pick this requirement "Choose Payment Option"? [YES]
160 U: YES
161 M: The requirement "Enter Payment Information" need to be evaluated. Would you like to choose the functional requirement "Enter Payment Information"? [YES, NO]
162 U: YES
163 M: Are you sure you want to pick this requirement "Enter Payment Information"? [YES]
164 U: YES
165 M: The group of functional requirements, which compose the requirement "Set Payment Information", have been pre-evaluated. [OK]
166 U: OK
167 M: The functional requirement "Set Payment Information" has been picked. [OK]
168 U: OK

Listing 4.8: Dialogue Utterances (Part 8)

169 M: The functional requirement "Choose Payment Option" has been picked. [OK]
170 U: OK
171 M: The functional requirement "Enter Payment Information" is going to be picked. [OK]
172 U: OK
173 M: A group of quality requirements, which are related with this requirement, are going to be pre-evaluated. [OK]
174 U: OK
175 M: The requirement "Average security for payment input" need to be evaluated. Would you like to choose the quality requirement "Average security for payment input"? [YES, NO]
176 U: NO
177 M: Are you sure you want to abandon this requirement "Average security for payment input"? [YES]
178 U: YES
179 M: The requirement "High security for payment input" need to be evaluated. Would you like to choose the quality requirement "High security for payment input"? [YES, NO]
180 U: YES
181 M: Are you sure you want to pick this requirement "High security for payment input"? [YES]
182 U: YES
183 M: The group of quality requirements which, are related with requirement "Enter Payment Information", have been pre-evaluated. [OK]
184 U: OK
185 M: The functional requirement "Enter Payment Information" has been picked. [OK]
186 U: OK
187 M: The quality requirement "Average security for payment input" has been abandoned. [OK]
188 U: OK
189 M: The quality requirement "High security for payment input" has been picked. [OK]
190 U: OK
4.5 Case Study

4.5 Case Study

M: The requirement "Platform Mobile" need to be evaluated. Would you like to choose the functional requirement "Platform Mobile"? [YES, NO]

U: YES

M: If you choose to pick the functional requirement "Platform Mobile", 9 requirements, which haven’t been evaluated, are prerequisite of this requirement. They will be picked. Are you sure you want to pick this requirement "Platform Mobile"? [YES]

U: YES

M: The functional requirement "Platform Mobile" has been picked. [OK]

U: OK

M: The functional requirement "Add a mobile payment option" is going to be picked. [OK]

U: OK

M: A group of quality requirements, which are related with this requirement, are going to be pre-evaluated. [OK]

U: OK

M: The requirement "Get Credit Card Information" need to be evaluated. Would you like to choose the quality requirement "Get Credit Card Information"? [YES, NO]

U: YES

M: If you choose to pick the quality requirement "Get Credit Card Information", 1 requirement, which hasn’t been evaluated, is contradicted with this requirement. It cannot be picked. 1 requirement, which hasn’t been evaluated, is prerequisite of this requirement. It will be picked. Are you sure you want to pick this requirement "Get Credit Card Information"? [YES]

U: YES

M: The group of quality requirements which, are related with requirement "Add a mobile payment option", have been pre-evaluated. [OK]

U: OK

Listing 4.9: Dialogue Utterances (Part 9)

M: The functional requirement "Add a mobile payment option" has been picked. [OK]

U: OK

M: The functional requirement "Delivery Information" has been picked. [OK]

U: OK

M: The functional requirement "Add product" has been picked. [OK]

U: OK

M: The functional requirement "Get Front Cover Image" has been picked. [OK]

U: OK

M: The functional requirement "Get Screen Size" has been picked. [OK]

U: OK

M: The functional requirement "Get Camera Device ID" has been picked. [OK]

U: OK

M: The functional requirement "Get GPS Coordinates" has been picked. [OK]

U: OK

M: The functional requirement "Get QR Code" has been picked. [OK]

U: OK

M: The functional requirement "Get Camera Device ID" has been picked. [OK]

U: OK

M: The functional requirement "Get GPS Device ID" has been picked. [OK]

U: OK

M: The quality requirement "Get Mobile Wallet" has been abandoned. [OK]

U: OK

M: The requirement "Manage Payment Information" need to be evaluated. Would you like to choose the functional requirement "Manage Payment Information"? [YES, NO]

U: YES

Listing 4.10: Dialogue Utterances (Part 10)
<table>
<thead>
<tr>
<th>Line</th>
<th>Dialogue Utterances</th>
</tr>
</thead>
<tbody>
<tr>
<td>231</td>
<td>M: If you choose to pick the functional requirement &quot;Manage Payment Information&quot;, 4 requirements, which haven't been evaluated, are prerequisite of this requirement. They will be picked. Are you sure you want to pick this requirement &quot;Manage Payment Information&quot;? [YES]</td>
</tr>
<tr>
<td>232</td>
<td>U: YES</td>
</tr>
<tr>
<td>233</td>
<td>M: The functional requirement &quot;Manage Payment Information&quot; has been picked. [OK]</td>
</tr>
<tr>
<td>234</td>
<td>U: OK</td>
</tr>
<tr>
<td>235</td>
<td>M: The functional requirement &quot;Get Reference to a Payment Option&quot; has been picked. [OK]</td>
</tr>
<tr>
<td>236</td>
<td>U: OK</td>
</tr>
<tr>
<td>237</td>
<td>M: The functional requirement &quot;Remove a Payment Option&quot; has been picked. [OK]</td>
</tr>
<tr>
<td>238</td>
<td>U: OK</td>
</tr>
<tr>
<td>239</td>
<td>M: The functional requirement &quot;Add a Payment Option&quot; has been picked. [OK]</td>
</tr>
<tr>
<td>240</td>
<td>U: OK</td>
</tr>
<tr>
<td>241</td>
<td>M: The functional requirement &quot;Pick a payment option&quot; has been picked. [OK]</td>
</tr>
<tr>
<td>242</td>
<td>U: OK</td>
</tr>
<tr>
<td>243</td>
<td>M: The functional requirement &quot;Get a list of payment options&quot; has been picked. [OK]</td>
</tr>
<tr>
<td>244</td>
<td>U: OK</td>
</tr>
<tr>
<td>245</td>
<td>M: All requirements have been evaluated. Would you like to make any change to your selections (e.g. add requirements, remove requirements)? [ADD, REMOVE, NO]</td>
</tr>
<tr>
<td>246</td>
<td>U: NO</td>
</tr>
<tr>
<td>247</td>
<td>M: Requirement elicitation has been completed. Thank you very much for using. [OK]</td>
</tr>
<tr>
<td>248</td>
<td>U: OK</td>
</tr>
</tbody>
</table>

Listing 4.11: Dialogue Utterances (Part 11)
The final list of picked requirements is displayed in Listing 4.12. Listing 4.13 shows the final list of abandoned requirements.

| 1 | Manage a shopping cart |
| 2 | Locate a book |
| 3 | Get reference to a book |
| 4 | Pick a book from a book list |
| 5 | Get a list of relevant books |
| 6 | Search relevant books |
| 7 | Sort books in a list |
| 8 | Get detailed info of a book |
| 9 | Get publication info |
| 10 | Get contents |
| 11 | Get reference to an item in the shopping cart |
| 12 | Remove a book from cart |
| 13 | Add a book to the shopping cart |
| 14 | Pick an item in shopping list |
| 15 | Get a shopping list |
| 16 | Finish the Order |
| 17 | Search in book keywords |
| 18 | Search |
| 19 | Advanced search |
| 20 | Exact match |
| 21 | Get the Order Summary |
| 22 | Calculate Total Price |
| 23 | Calculate Book Price |
| 24 | Calculate Service Price |
| 25 | Set Delivery Information |
| 26 | Choose Delivery Option |
| 27 | Enter Delivery Information |
| 28 | Set Payment Information |
| 29 | Choose Payment Option |
| 30 | Enter Payment Information |
| 31 | High security for payment input |
| 32 | Platform Mobile |
| 33 | Add a mobile payment option |
| 34 | Delivery Information |
| 35 | Add product |
| 36 | Get Front Cover Image |
| 37 | Get Screen Size |
| 38 | Get GPS Coordinates |
| 39 | Get QR Code |
| 40 | Get Camera Device ID |
| 41 | Get GPS Device ID |
| 42 | Get Credit Card Information |
| 43 | Manage Payment Information |
| 44 | Get Reference to a Payment Option |
| 45 | Remove a Payment Option |
| 46 | Add a Payment Option |
| 47 | Pick a payment option |
| 48 | Get a list of payment options |

Listing 4.12: Picked Requirements

| 1 | Broad match |
| 2 | Average security for payment input |
| 3 | Get Mobile Wallet |

Listing 4.13: Abandoned Requirements
Listing 4.14 shows the output of the OC Viewer. It lists all the ontology combinations that occurred during the session. Line 2 displays the output of the combination with `Search` ontology, $O_i$. The combination with `OrderSummary` ontology, $O_j$, is displayed in Line 12. Line 20 shows the output of the combination with `PlatformMobile` ontology, $O_{Mobile}$. The output of the combination with `ManagePaymentInfo` ontology, $O_k$, is shown in Line 24.

```
1  "Search relevant books" is a leaf node
2       combine successful! (Search relevant books, Search, ISA, 0.0, ./KnowledgeBase/Ontology/Search.owl)
3  "Get publication info" is a leaf node
4       combine unsuccessful! No correspondence found!
5  "Get contents" is a leaf node
6       combine unsuccessful! No correspondence found!
7  "Remove a book from cart" is a leaf node
8       combine unsuccessful! No correspondence found!
9  "Pick an item in shopping list" is a leaf node
10       combine unsuccessful! No correspondence found!
11  "Finish the Order" is a leaf node
12       combine successful! (Finish the Order, Get the Order Summary, TYPEOF, 0.0, ./KnowledgeBase/Ontology/OrderSummary.owl)
13  "Advanced search" is a leaf node
14       combine unsuccessful! No correspondence found!
15  "Calculate Book Price" is a leaf node
16       combine unsuccessful! No correspondence found!
17  "Calculate Service Price" is a leaf node
18       combine unsuccessful! No correspondence found!
19  "Choose Delivery Option" is a leaf node
20       combine successful! with mobile platform ontology (Choose Delivery Option, Delivery Information, ISA, 0.0, KnowledgeBase/OntologyMobile/PlatformMobile.owl)
21  "Enter Delivery Information" is a leaf node
22       combine unsuccessful! No correspondence found!
23  "Choose Payment Option" is a leaf node
24       combine unsuccessful! No correspondence found!
25  "Get Screen Size" is a leaf node
26       combine unsuccessful! No correspondence found!
27  "Get Camera Device ID" is a leaf node
28       combine unsuccessful! No correspondence found!
29  "Get GPS Device ID" is a leaf node
30       combine unsuccessful! No correspondence found!
31  "Remove a Payment Option" is a leaf node
32       combine unsuccessful! No correspondence found!
33  "Add a Payment Option" is a leaf node
34       combine unsuccessful! No correspondence found!
35  "Get a list of payment options" is a leaf node
36       combine unsuccessful! No correspondence found!
```

Listing 4.14: Entire OC Viewer output
The interactive RE session output is generated using the requirement evaluation process [75] mentioned in Section 2.4. Through the use of ontology combinations, platform-dependent requirements are included and evaluated. The requirement evaluation algorithm merges the various requirements such as AddMobilePaymentOption, DeliveryInfo, GetFrontCoverImage into the ManageShoppingCart requirement, taking into account the inputs and outputs of all these requirements. The output OWL-S document is presented in Listing 4.15. The platform-dependent details are highlighted in gray. The documents, BookShoppingProcess.owl and BookShoppingQuality.owl, are imported by the profile document and define the instances of inputs, outputs and qualities [75]. By using this OWL-S description, services can be discovered by semantic capability matching. Then the services can be composed and executed based on the corresponding service composition information offered by the service providers [75].

```xml
<?xml version="1.0" encoding="UTF-8"?>
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
xmlns:profile="http://www.daml.org/services/owl-s/1.2/Profile.owl"
xmlns:owl="http://www.w3.org/2002/07/owl#"
<owl:Ontology rdf:about=""/>
<owl:imports rdf:resource="http://www.daml.org/services/owl-s/1.2/Profile.owl"/>
<owl:imports rdf:resource="http://www.w3.org/1999/02/22-rdf-syntax-ns#"/>
</owl:Ontology>

<profile:Profile rdf:ID="Manage_a_shopping_cart">
  <profile:textDescription>Manage a shopping cart</profile:textDescription>
  <profile:hasProcess rdf:resource="http://www.semanticweb.org/ontologies/InteractiveRE/BookShoppingProcess.owl#EnterNameInfo"/>
</profile:Profile>
```
Listing 4.15: OWL-S output file with platform-dependent details highlighted in gray
4.6 Contributions

4.6.1 Overview

This section covers the contributions of the study undertaken for this thesis.

4.6.2 Enhanced Interactive Requirements Elicitation

The Ontology Combination methodology [71][72], as shown in this thesis, provides a seamless interactive experience for acquiring requirements from multiple ontologies. Ontology Combinations are invisible to the user as they happen in the background and without the user’s knowledge. The dynamic nature of Ontology Combination, in the context of interactive RE, is shown in an abstract manner by Figure 4.19.

Ontologies that are being combined are assumed to be distinct, i.e., a concept to be matched only exists in the primary SPL ontology, $O_{SPL}$ as an ‘imprecise’ concept, and ontologies, $O_i$, $O_j$, $O_k$, and $O_{Mobile}$, taken from the RE Knowledge base, $K_{RE}$, serve to refine that concept. This also illustrates the advantages of Ontology Combination over ontology merging. Instead of creating a merged ontology, which would require more...
resources in terms of time and memory, combining ontologies is less resource-intensive. Iterating over the Ontology Combination process, requirements can be fine-tuned to the smallest level of detail that the ontologies provide in $K_{RE}$.

Experiments were conducted based on the number of ontologies being combined together during interactive requirements elicitation sessions. The desired outcome was logical combinations between ontologies being combined together in a responsive manner, transparent to the user. The requirements specification generated is ‘richer’ after the combination process as compared to the specifications generated before the combination occurred. In Section 4.4.2, there were sixteen requirements in the primary SPL ontology $O_{SPL}$ - by combining $O_{SPL}$ with $O_i$, $O_j$, $O_k$ and $O_{Mobile}$, the number of requirements were tripled to forty-eight requirements. This is a significant increase over the initial ontology in the potential of acquiring requirements.

### 4.6.3 Extending Customization to Mobile Applications

The previous interactive RE system for SOA-based SPL produced specifications for traditional (desktop-based) applications [75]. This thesis has extended that approach to produce specifications for generating SOA applications for mobile operating systems. A use-case was given in Section 4.4.3 Scenario III - Ontology of Mobile SOA Functions. Contradictions due to platform-dependent features were resolved through the Ontology Combination process. As the ecosystem of mobile software and hardware changes rapidly, this will enable applications to be built for specific versions of particular mobile operating systems in a rapid manner.
Chapter 5

Conclusion and Future Directions

5.1 Overview

This chapter concludes the thesis and presents some potential directions of future research.

5.1.1 Conclusion

The methodology of Ontology Combination has been proposed in this thesis. The Combine algorithm brings ontologies together at run-time, dynamically enhancing the interactive RE process. The Scenarios presented in this thesis illustrate the effectiveness of the Ontology Combination methodology. Using this approach, interactive RE can also be used seamlessly for the purpose of customizing software for specific platforms, thereby helping to automate SPLs.
5.1.2 Future Directions

Run-time application of Ontology Combinations provides multiple future directions of further research. The Ontology Combination methodology presented in this work uses ontologies that are derived from the ontology requirement meta-model and performs combination on these ontologies. A future direction of research can look into combining ontologies that follow different designs, where the primary focus would be on trying to accommodate the overlap and contradictions that the set of axioms from different ontologies would entail.

Also, the Ontology Combination approach can be expanded by realizing a complete SPL framework for SOA based applications. This framework can potentially create applications targeting multiple platforms, with features developed and maintained to suit different hardware specifications. A comprehensive SPL framework can be extended to both software and hardware feature resolution and creation of SOA software that synthesizes the knowledge of both software and hardware features.
Bibliography


# Vita Auctoris

<table>
<thead>
<tr>
<th>NAME</th>
<th>Shubhrendu Tripathi</th>
</tr>
</thead>
<tbody>
<tr>
<td>PLACE OF BIRTH</td>
<td>New Delhi, India</td>
</tr>
<tr>
<td>YEAR OF BIRTH</td>
<td>1983</td>
</tr>
<tr>
<td>EDUCATION</td>
<td>Memorial High School, Houston, Texas, USA</td>
</tr>
<tr>
<td></td>
<td>1998-1999</td>
</tr>
<tr>
<td></td>
<td>B.Sc., University of Houston, Texas, USA</td>
</tr>
<tr>
<td></td>
<td>1999-2002</td>
</tr>
</tbody>
</table>