



Norwegian University of
Science and Technology

A Demonstration on Service Compositions based on Natural Language Request and User Contexts

Kirati Sutthikulphanich

Master of Science in Communication Technology

Submission date: June 2008

Supervisor: Rolv Bræk, ITEM

Co-supervisor: Mazen Shiaa, Malek, ITEM

Hubert Baumeister, DTU, Denmark

Problem Description

The task is to discover and compose services for the end-user based on natural language requests. The main idea is to semantically analyze the request, highlight keywords that will be linked to concepts in certain ontologies. These concepts will both be related to service components and context, mainly user context. As the title suggests, the task will encompass three main phases the access, discovery and composition of the end-user services.

The phases of discovery and composition will be studied first. The student will work with the SPICE platform functionality for service discovery and composition. The student will also work with the SPICE context handling platform that takes care of user's subscriptions, profile, preferences, location, terminal, etc. in this regard the student will build together an end-user studio for service composition, preferably working on different end-user devices.

The service access phase will be dealt with based on the progress performed on the other two phases. As an initial and compulsory part of this phase user Authentication and Authorization procedures must be carried out. The student will look into the work on security issues within the SPICE project. There exist mechanisms to handle the user login, registration, etc. These mechanisms will be studied and partially developed as an extension to the end-user.

Assignment given: 15. January 2008
Supervisor: Rolv Bræk, ITEM

Abstract

The significant growth in mobile markets exposes business niches [1]. A number of heterogeneous services have been offered to end users by several service providers. The end users can fully benefit from wide varieties of services only when these heterogeneous services can be composed together (i.e. service composition is created).

This fact motivates an invention of frameworks that support diverse features such as service creations and compositions. The frameworks aid new market players to enter and offer innovative services. Such frameworks could benefit users by offering a big pool of services, and service providers by simplifying deployment process in adopting groundbreaking services [2].

To achieve this, **SPICE (Service Platform for Innovative Communication Environment)** project has been established. In SPICE project, the researchers have developed, among other things, two environments: a **Service Creation Environment (SCE)** and a **Service Execution Environment (SEE)** [3]. An objective of SCE is to create and compose web services and Telecommunication services. An objective of SEE is to execute compositions of the services created by SCE. Main component developed in SCE is **Automatic Composition Engine (ACE)**. ACE dynamically constructs service composition that satisfies a request of service developers. The request has to be constructed in a formal way. Therefore, the informal request such as natural language request (i.e. in English) needs to be transformed into a formal service composition request before it is sent to the ACE. Thus, clients of ACE are only limited to service developers who have enough knowledge to construct the formal request.

To bring SCE out to the market where end users do not possess much of technical knowledge, this thesis proposes a development of **Intelligence Agent (IA)** as a new component of SCE. IA construct service composition from an informal request in natural language by using **Natural Language Processing (NLP)** technique.

A context handling feature has also been considered in this thesis. In SPICE project, the researches have developed **Knowledge Management Framework (KMF)** that supports context handling. This thesis demonstrates a possibility to integrate IA with KMF so as to improve SCE.

As a result of this thesis, we have developed a prototype of IA. IA receives a request from end user in natural language (English), analyze the request, search and compose for service composition candidates that satisfy the request and deploy the candidates into an artificial language, SPATEL [43]. We have also developed a module of IA that registers and subscribes for a user context published by a knowledge source in KMF framework. IA takes into account a context of user (e.g. user location, user authentication) when it selects services from the service repository. IA also ensures that the desired service composition will be deployed when certain condition on user context arises.

Preface

This master thesis is written at the Department of Telematics at the Norwegian University of Science and Technology (NTNU) in the spring semester of 2008. The master thesis is submitted to fulfill graduation requirement of Master of Science in Security and Mobile Computing taken place at Norwegian University of Science and Technology (NTNU) and Technical University of Denmark (DTU).

First of all, I would like to use this opportunity to thank my academic supervisors at the Department of Telematics, Norwegian University of Science and Technology (NTNU) as well as my academic supervisor at the Department of Informatics and Mathematical Modeling, Technical University of Denmark (DTU). My supervisors always give me helpful supports and invaluable advices during the six-month period of the thesis. Professor Rolv Braek (NTNU) and Associate Professor Hubert Baumeister (DTU) have given me very helpful comments on my writing. Mazen Malek Shiaa (NTNU) has been my main technical adviser. I strongly believe that his invaluable advices have significantly enhanced the quality of my work.

Furthermore, I would like to express my gratitude to Professor Lars Hellan at the Department of Linguistics, Norwegian University of Science and Technology (NTNU) for his constant support on natural language processing.

Another acknowledgement would go to people in Service Platform for Innovative Communication Environment (SPICE) project. In particular, I would like to thank Benoit Thiell for giving me very helpful explanations on Automatic Composition Engine (ACE) and SPICE Advance Service Description Language for Telecommunication Services (SPATEL).

I would also like to show my appreciation to the NordSecMob consortium. The NordSecMob consortium gave me an opportunity to study in this beautiful continent. This is an experience that I will never forget. Special thank goes to Mona Nordaune (NTNU) and Han Gao (DTU), my program coordinators, for their kind helps during my stay in Europe.

I would also like to thank all people who have denoted their time to proofread my thesis.

Last but not least, I would like to thank my family and my friends for their supports. In particular, I would like to thank my parents who motivated me to study software engineering, in the first place.

Trondheim, June 2008

Kirati Sutthikulphanich

Table of Content

Abstract.....	iv
Preface.....	vi
Table of Content	viii
List of Figures	xii
List of Tables	xvi
List of Abbreviation.....	xix
Chapter 1.....	1
Introduction.....	1
1.1 Background and Motivation	2
1.2 Architecture Overview.....	6
1.3 IA components.....	10
1.4 Thesis Scope and Limitations	13
1.5 Research Methodology	13
1.6 Thesis Outline	16
Chapter 2.....	17
Related technologies	17
2.1 Natural Language Processing (NLP)	17
2.1.1 Parsing.....	18
2.1.2 Information Retrieval.....	21
2.1.3 Semantic Analysis.....	22
2.1.4 Semantic Representation.....	23
2.1.5 Minimal Recursion Semantic (MRS).....	28
2.1.6 Related works in service composition based on NLP.....	31
2.2 Semantic Web Technologies.....	32
2.2.1 Resource Description Framework (RDF)	33
2.2.2 The Web Ontology Language (OWL)	35
2.2.3 A Semantic Web Framework for Java (Jena)	38
2.3 Service Creation Environment (SCE).....	42
2.4 Knowledge Management Framework (KMF)	48
2.4.1 Knowledge Management Framework (KMF): Overview.....	49
2.4.2 Knowledge Source	52
2.4.3 Knowledge Sink.....	53
2.4.4 Knowledge Broker	54
2.5 Security	55
2.5.1 Entity Authentication and Authorization	55
Chapter 3.....	59
Requirement Analysis and Design.....	59
3.1 Intelligent Agent (IA) Requirement Analysis.....	59
3.2 IA environment	61
3.3 Overall design	62
3.4 Design of IA repositories	65
3.4.1 Ontology construction approach	66
3.4.2 Service Repository	67
3.4.3 Goal Ontology.....	68
3.4.4 Input/Output ontology.....	70

3.4.5	Synonym ontology	71
3.4.6	Recommended Service Rule	72
3.4.7	Restriction Service Rule	73
3.4.8	Dictionary file	73
3.4.9	Minimal Recursive Semantic (MRS) file	74
3.5	Design of functionalities	78
3.5.1	Easy-to-use interface.....	78
3.5.2	IA Performance	79
3.5.3	Information Extraction Capability	81
3.5.4	Service Discovery, Service Weighting and Service Filtering.....	82
3.5.5	Service Organization and Service Ranking	84
3.5.6	Service Parameter Extraction.....	85
3.5.7	ACE-Interface Capability	85
3.5.8	NLP-interface.....	86
3.5.9	KMF Interface.....	89
3.5.10	Context Subscription.....	89
3.5.11	Context Retrieval	89
3.5.12	Context Understanding	89
3.5.13	Context Handling	90
3.5.14	Synchronization	90
3.5.15	Authentication/Authorization Retrieval Capability	90
3.5.16	Authentication/Authorization Handling Capability.....	91
Chapter 4	93
Implementation	93
4.1	UML diagram and classes information.....	93
4.2	IA Data Structure	101
4.2.1	Data Structure of Service Repository	102
4.2.2	Data Structure of Goal, Input/Output, Synonym Ontology	102
4.2.3	Data Structure of Recommended Service Rule Table	104
4.2.4	Keyword table, Goal-Weight table, Keyword-Weight table and Common Goal table	104
4.2.5	Service-Goal Pair	105
4.2.6	Service Composition Candidates	106
4.2.7	Data Structures for manipulating NLP result.....	107
4.3	IA main process	108
Chapter 5	111
Result and Discussion	111
5.1	Three user scenarios: The results.....	111
5.1.1	Scenario 1.....	112
5.1.1.1	Result of scenario 1: Service compositions in SPATEL	113
5.1.1.2	Result of scenario 1: Service input parameters.....	117
5.1.2	Scenario 2.....	118
5.1.2.1	Result of scenario 2: Service compositions in SPATEL	118
5.1.2.2	Result of scenario 2: Service input parameters.....	122
5.1.3	Scenario 3.....	124
5.1.3.1	Result of scenario 3: Service compositions in SPATEL	125
5.1.3.2	Result of scenario 3: Service input parameters.....	129
5.2	IA Correctness	130
5.3	Performance Measurement	142
5.4	Discussion on performance: User Scenarios.....	143

5.4.1	Format of the result.....	143
5.4.2	Discussion on performance: User Scenario 1	144
5.4.3	Discussion on performance: Scenario 2.....	150
5.4.4	Discussion on performance: Scenario 3.....	157
5.5	A Performance Improvement with MRS	164
Chapter 6	167
Conclusion and Future work	167
6.1	Conclusion	167
6.2	Future work.....	170
Appendix A	173
IA Algorithm and APIs	173
A.1	Algorithm: Service Repository Interpretation	173
A.2	Algorithm: Word Extraction.....	174
A.3	Algorithm: Service Distance.....	175
A.4	Algorithm: Service Connections.....	176
A.5	Algorithm: Service Composition	178
A.6	Algorithm: Sequence of Event.....	179
A.7	APIs.....	180
A.7.1	SWT Designer.....	181
A.7.2	Jena API.....	181
A.7.3	OpenNLP API.....	183
A.7.4	KMF API	184
A.7.5	DOM API.....	186
Appendix B	189
Problems during implementation	189
B.1	The First Modification on Project Schedule	189
B.2	The Second Modification on Project Schedule.....	189
B.3	The Third Modification on Project Schedule.....	190
Appendix C	191
IA configuration files	191
C.1	Service Repository (services.csv)	191
C.2	Goal Ontology (Goals.owl).....	194
C.3	Input/Output Ontology (IO.owl).....	201
C.4	Synonym Ontology (Synonym.owl)	209
C.5	Recommended Service Rule (ContextRule.txt).....	210
C.6	Restriction Service Rule (RestrictionRule.txt)	211
C.7	Dictionary file (norwegian2english.txt).....	211
C.8	MRS for “Find traffic in Paris and send them by email” (nlp.xml).....	211
C.9	MRS for “Find vegetarian restaurant in my area and send it by sms or email”(nlp2.xml).....	216
C.10	MRS for “Find weather report in Paris, translated to English and send it by email”(nlp3.xml).....	222
C.11	MRS for “Search a flight from London to Paris on 10-06-2008 and book it”(nlp5.xml).....	228
C.12	MRS for “Find an address of theater in my area and send it to my email”(nlp4.xml).....	233
Appendix D	241
User Manual	241
Appendix E	247
Hardware and Software Specification	247

Appendix F.....	249
IA Directory Structure	249
Appendix G.....	251
StopWatch.java	251
Glossary	253
References.....	261

List of Figures

Figure 1: SCE and SEE architecture	4
Figure 2: KMF architecture	5
Figure 3: Our proposed architecture	5
Figure 4: Service Repository file	7
Figure 5: An example of ontology class and its relation with another ontology class in goal ontology	7
Figure 6: An example of ontology class and its relation with another ontology class in input/output ontology	8
Figure 7: An example of ontology class and its relation with another ontology class in synonym ontology	8
Figure 8: An example of recommended service rule	8
Figure 9: An example of restriction service rule	8
Figure 10: An example of dictionary file	8
Figure 11: An example of XML node in MRS	8
Figure 12: IA internal components	12
Figure 13: Division of works	15
Figure 14: The correct parse tree for the sentence “Book that flight” according to the grammar in Figure 15 [4]	19
Figure 15: A miniature English grammar and lexicon [4]	19
Figure 16: An example of parse trees generated by top-down approach	19
Figure 17: An example of parse tree generated by bottom-up approach	20
Figure 18: A simple pipeline approach to semantic analysis	22
Figure 19: The semantic representation using FOPC	26
Figure 20: A semantic representation of “every big white horse sleep” based on recursive structure approach	30
Figure 21: An abstract view on a semantic representation of “every big white horse sleep” based on MRS	30
Figure 22: A semantic representation of “every big white horse sleep” based on MRS	30
Figure 23: A semantic representation of “every dog chase some white cat” based on MRS	31
Figure 24: An example of RDF document	33
Figure 25: An example of Collection Attribute	34
Figure 26: An example of RDF container	35
Figure 27: An example of OWL concept definition	36
Figure 28: An example of ontology class and its relation with another ontology class in goal ontology	37
Figure 29: An example of ontology class and its relation with another ontology class in input/output ontology	37
Figure 30: An example of ontology class and its relation with another ontology class in synonym ontology	38
Figure 31: An example of RDF graph	40
Figure 32: The Statement seen by the OntModel	40
Figure 33: Syntax of formal request sent to ACE	44
Figure 34: An example of formal composition request	45

Figure 35: An example of SPATEL (1).....	47
Figure 36: An example of SPATEL (2).....	48
Figure 37: An example of location context.....	50
Figure 38: An example of authentication context.....	51
Figure 39: An example of restriction context	51
Figure 40: Types of Knowledge Source (by functionality)	53
Figure 41: An interaction diagram between IA and ACE.....	62
Figure 42: A sequence diagram of IA components (1).....	63
Figure 43: A sequence diagram of IA components (2).....	63
Figure 44: A sequence diagram of IA components (3).....	63
Figure 45: A sequence diagram of IA components (4).....	64
Figure 46: A sequence diagram of IA components (5).....	64
Figure 47: A structure of Service Repository	68
Figure 48: An example of Goal Ontology	69
Figure 49: A hierarchy of goal ontology concept, GetBusinessInfo	70
Figure 50: An example of Input/Output Ontology	71
Figure 51: An example of synonym ontology	72
Figure 52: An example of recommended service rule	73
Figure 53: An example of restriction service rule (RestrictionRule.txt)	73
Figure 54: An example of dictionary file (norwegian2english.txt)	74
Figure 55: An example of a result of MRS analysis on user request, “Finn trafikkforholdene i Paris og send dem pr. email”	77
Figure 56: MRS corresponding to user request “Finn trafikkforholdene i Paris og send dem pr. email” (1).....	77
Figure 57: MRS corresponding to user request “Finn trafikkforholdene i Paris og send dem pr. email” (2).....	78
Figure 58: IA graphical interface	79
Figure 59: IA part of speech tagging	81
Figure 60: A result of running IA attached with ACE functionality	86
Figure 61: An example of SPATEL exported by ACE.....	88
Figure 62: UML diagram of IA	94
Figure 63: UML diagram of UIParser and dependent classes (1)	96
Figure 64: UML diagram of UIParser and dependent classes (2)	97
Figure 65: UML diagram of UIParser and dependent classes (3)	98
Figure 66: UML diagram of UIParser and dependent classes (4)	99
Figure 67: UML diagram of UIParser and dependent classes (5)	100
Figure 68: UML diagram of xmlParser and dependent classes	101
Figure 69: UML diagram of UI and dependent classes	101
Figure 70: A declaration on data structure of service repository	102
Figure 71: A declaration on an entry in service repository.....	102
Figure 72: Data Structures for Ontology	103
Figure 73: Data Structure for Service Rule Table	104
Figure 74: A declaration on an entry in Recommended Service Rule.....	104
Figure 75: Four tables that are necessary for service discovery	105
Figure 76: Duplication of service name in service repository	106
Figure 77: Duplication of service goal in service repository	106
Figure 78: Data structure that utilizes Service-Goal pair.....	106
Figure 79: Fields of InputPair	106
Figure 80: A declaration of SC fields	107
Figure 81: A declaration of ServiceOperation fields	107

Figure 82: A declaration of Parameter fields.....	107
Figure 83: Data Structure for manipulating NLP result.....	108
Figure 84: Field declarations of predicate object.....	108
Figure 85: Recommended Service Rule for scenario 1	113
Figure 86: A SPATEL file generated from a request “Find vegetarian restaurant in my area and send it by sms or email”	115
Figure 87: A SPATEL file generated when a user location is changed to street@telin.nl.....	116
Figure 88: Service input parameters extracted from a request “Find vegetarian restaurant in my area and send it by sms or email” (IA without MRS).....	118
Figure 89: Service input parameters extracted from a request “Find vegetarian restaurant in my area and send it by sms or email” (IA with MRS).....	118
Figure 90: A SPATEL file generated from a request “Find weather report in Paris, translated to English and send it by email”.....	121
Figure 91: A SPATEL file generated from a request “Search a flight from London to Paris on 10-06-2008 and book it”	121
Figure 92: Service input parameters extracted from a request “Find weather report in Paris, translated to English and send it by email” (IA without MRS).....	123
Figure 93: Service input parameters extracted from a request “Find weather report in Paris, translated to English and send it by email” (IA with MRS)	123
Figure 94: Service input parameters extracted from a request “Search a flight from London to Paris on 10-06-2008 and book it” (IA without MRS).....	124
Figure 95: Service input parameters extracted from a request “Search a flight from London to Paris on 10-06-2008 and book it” (IA with MRS).....	124
Figure 96: Recommended Service Rule for scenario 3	125
Figure 97: Recommended Service Rule for scenario 3 (continue).....	125
Figure 98: A SPATEL file generated from a request “Find an address of theater in my area and send it to my email”.....	127
Figure 99: A SPATEL file generated when a user location is changed to customer_siteE@telin.nl.....	128
Figure 100: Service input parameters extracted from a request “Find an address of theater in my area and send it to my email” (IA without MRS).....	130
Figure 101: Service input parameters extracted from a request “Find an address of theater in my area and send it to my email” (IA with MRS).....	130
Figure 102: IA Recommended Service Rule	131
Figure 103: IA Restriction Service Rule.....	131
Figure 104: All keywords found in goal ontology.....	131
Figure 105: IA builds semantic-distance table.....	132
Figure 106: IA extracts keywords from service goals	132
Figure 107: IA calculates a weight for each keyword	133
Figure 108: IA matches input keywords with service goals.....	134
Figure 109: IA constructs a list of consolidated services (corresponding to service goals).....	134
Figure 110: IA maps instances of service input parameters to corresponding services	135
Figure 111: IA constructs a list of service composition candidates.....	136
Figure 112: IA ranks a list of service composition candidates	136
Figure 113: IA ranks a list of service composition candidates (2)	137
Figure 114: IA ranks a list of service composition candidates based on MRS.....	138
Figure 115: IA invokes a service composition defined in ContextRule.txt.....	139

Figure 116: IA restricts access to services as defined in RestrictionRule.txt	139
Figure 117: IA restricts an access to service, GlobalSMS as response to a change in restriction context.....	140
Figure 118: IA allows an access to service, GlobalSMS as response to a change in authentication context.....	140
Figure 119: A user location is changed to A2.05@telin.nl for user, Martin.Wibbels@telin.nl	141
Figure 120: IA ignores the context that belongs to different users.....	141
Figure 121: A mapping between service and its input parameters derived from user context.....	142
Figure 122: List of Composition Candidates for scenario 1 (MRS disabled).....	147
Figure 123: List of Composition Candidates for scenario 1 (MRS enabled)	149
Figure 124: IA constructs a service composition when the user location has been changed to street@telin.nl.....	150
Figure 125: IA constructs a service composition when the user location has been changed to restaurant@telin.nl	150
Figure 126: List of Composition Candidates for scenario 1 (MRS disabled).....	153
Figure 127: List of Composition Candidates for scenario 1 (MRS enabled)	154
Figure 128: List of Composition Candidates for scenario 2-2 (MRS disabled)	156
Figure 129: List of Composition Candidates for scenario 2-2 (MRS enabled)	157
Figure 130: List of Composition Candidates for scenario 3 (MRS disabled).....	160
Figure 131: List of Composition Candidates for scenario 2 (MRS enabled)	163
Figure 132: IA constructs a service composition when a user context has been changed to customer_siteA@telin.nl	164
Figure 133: IA constructs a service composition when a user context has been changed to customer_siteE@telin.nl.....	164

List of Tables

Table 1: A relationship between properties and repository files	66
Table 2: Word-level tag [33].....	82
Table 3: All user requests in English and Norwegian.....	112

List of Abbreviation

Abbreviation	Meaning
API	Application Programming Interface
ACE	Automatic Composition Engine
BPEL	Business Process Execution Language
CMF	Context Management Framework
CA	Certification Authority
CLM ⁺	Casual Link Matrix
CFG	Context Free Grammar
EP	Elementary Predication
FSA	Finite State Automata
FOPC	First Order Predicate Calculus
GUI	Graphical User Interface
HPSG	Head-Driven Phrase Structure Grammar
IA	Intelligent Agent
JVM	Java Virtual Machine
KMF	Knowledge Management Framework
MRS	Minimal Recursive Semantic
NLP	Natural Language Processing
NF	Non-functional property
OWL	Web Ontology Language
OOP	Object-oriented programming language
QoC	Quality of Information Measure
QoS	Quality of Service
RDF	Resource Description Framework
RPC	Remote Procedure Call
SCE	Service Creation Environment
SEE	Service Execution Environment
SOAP	Simple Object Access Protocol
SWT	Standard Widget Toolkit
SPICE	Service Platform for Innovative Communication Environment
SPATEL	SPICE Advance Service Description Language for Telecommunication Services
UI	User Interface
URI	Uniform Resource Description
UDDI	Universal Description, Discovery and Integration
WSDL	Web Service Definition Language
XML	Extensible Markup Language

Chapter 1

Introduction

Over half of the world's population will have a mobile phone by 2008, according to a new report from a United Kingdom-based research firm, Portio Research [1]. This indicates a significance growth in mobile markets. The significance growth in mobile markets exposes business niches. A number of heterogeneous services have been offered to end users by each service provider. The end users can fully benefit from wide varieties of services only when these heterogeneous services can be composed together (i.e. service composition is created).

This fact motivates an invention of frameworks that support diverse features such as service creations and compositions. The frameworks aid new market players to enter and offer innovative services. Such frameworks could benefit users by offering a big pool of services, and service providers by simplifying deployment process in adopting groundbreaking services [2].

To achieve this, **Service Platform for Innovative Communication Environment (SPICE)** project has been established. In SPICE project, the researchers have developed, among other things, two environments: a **Service Creation Environment (SCE)** and a **Service Execution Environment (SEE)** [3]. An objective of SCE is to create and compose web services and Telecommunication services. An objective of SEE is to execute compositions of the services created by SCE. Hence, one of the goals of the SPICE SCE is to facilitate service compositions based on different kinds of semantic annotations such as functional properties (e.g. service input, output and goal) and non-functional properties (e.g. service qualities). Main component developed in SCE is **Automatic Composition Engine (ACE)**. ACE dynamically constructs service composition that satisfies a request of service developers. The request has to be constructed in a formal way. Therefore, the informal request such as natural language request (i.e. in English) needs to be transformed into a formal service composition request before it is sent to the ACE. Specifically, the service descriptions need to be semantically annotated before the descriptions can be used to construct the formal request. Therefore, clients of ACE are only limited to service developers who have enough knowledge to construct the formal request.

To bring SCE out to the market where end users do not pose much of technical knowledge, this thesis proposes a development of **Intelligence Agent (IA)**. IA construct service composition from an informal request in natural language by using **Natural Language Processing (NLP)** technique.

It is worth noting that there was a similar work [3] (related to NLP) in SPICE project. Although we take the work [3] as our inspiration, our work uses different approach for NLP. This is because the work [3] is limited to bounded domains as it depends on a presumption on a structure of sentence (i.e. user request) and a number of services that are available.

This thesis also demonstrates how the user can benefit from context-aware application. For example, the application could deduce from user contexts what services a user might need and propose services to the user without user input [7]. For instance, the system proposes a list of news sources to the user when it infers from a context that the user has moved to a café assuming that user may tempt to read newspaper during his coffee break. Furthermore, authentication and authorization rules can be enforced based on derived user contexts (i.e. user privileges). In SPICE project, the researches have developed **Knowledge Management Framework (KMF)** that supports context handling. KMF comprises of two components, namely context source and context sink. For our demonstration purpose, we have integrated (i.e. implemented) a context sink module that has functionality in subscribing and retrieving user contexts with IA. In this thesis, we will handle two types of user contexts. These are location context and security context.

As for a result of this thesis, a prototype of IA has been developed. IA provides two functionalities, namely, service discovery and service composition. IA conducts service discovery and service composition based on an analysis of user request (using NLP) and an analysis of user context. IA can be served as a plug-in to SCE. Thus, SCE that is integrated with IA could support a request in natural language and handle a change in user context.

1.1 Background and Motivation

SPICE project has been established with 23 partners in 11 countries. One of the aims of SPICE project is “to build a user-transparent infrastructure that hides the complexity of services and applications crossing over different access domains and copes with the various access network technologies and offering a diversity of services” [2]. To achieve this objective, several frameworks have been created. Among of them are SCE and SEE. An objective of SCE is to create and compose web services and Telecommunication services. An objective of SEE is to execute compositions of the services created by SCE. Furthermore, KMF has also been created to help service providers in monitoring user contexts.

SCE and SEE together with their relations are shown in Figure 1. SCE provides service composition mechanism to service developers. SEE provides service execution mechanism for service developers. Basically, service developer submits a formal service composition request to ACE. ACE composes services and generates **SPATEL (SPICE Advanced Service Description Language for Telecommunication Services)** file describing the service composition. In fact, there could be many service compositions (and related SPATEL files) generated by ACE. In this case, it is up to the service developer to manually select the service composition that is most appropriate to his request. To execute a service composition, SEE is utilized. Service developer, at this stage, manually put SPATEL file obtained from ACE and related service input

Introduction

parameters to SPATEL execution engine. The SPATEL execution engine invokes the desired service components and return output to the user as a result. In Figure 1, we have shown that ACE inspects two types of files, namely, service repository and ontology. Service repository keeps services that are available to the user. Ontology keeps a semantic of services (e.g. what is the relation between service inputs and service outputs) in the repository. Essentially, these files store useful information for service composition.

KMF is shown in Figure 2. KMF helps service providers in monitoring user contexts. Components in KMF are classified into one of the following types: knowledge source, knowledge sink and knowledge broker. Knowledge source (e.g. a sensor) publishes user contexts. Knowledge sink subscribes user contexts. There could be many types of contexts that knowledge source can publish. In this thesis, we focus on two types of context. These are location context and security context. Basically, the KMF components interact with each other as follows:

- Knowledge source registers user contexts, that it is going to publish, with knowledge broker.
- Knowledge sink enquire knowledge broker about the knowledge source, that publishes desired user context.
- Knowledge sink subscribes with that knowledge source to obtain the contexts.

It is worth noting that components in KMF share common ontology file (context ontology) so as to agree on meanings of user contexts. It is also worth mentioning that, in our scenario, the knowledge source does not receive an input from external sources or sensors. Rather, the knowledge source, itself, is a sensor that can detect a context such as user location. We also assume, for simplicity, that service operator uses the knowledge source to establish security context.

One shortcoming of SCE is that the service composition request has to be constructed in a formal way. Therefore, the informal request such as natural language request (i.e. in English) needs to be transformed into a formal service composition request before it is sent to the ACE. Consequently, clients of ACE are only limited to service developers who have enough knowledge to construct the formal request. This thesis aims to demonstrate a possibility of SCE improvement with an integration of the intelligent agent (IA). With IA, the user does not require to pose technical knowledge as he can request for service compositions in natural language (i.e. English).

This thesis also aims to demonstrate a possibility of SCE improvement with KMF (i.e. to integrate SCE with KMF). With KMF, the service composition is more accurate as the user context is taken into account (e.g. service components, that are not suitable for the certain user context, will not be used for service composition). Moreover, security rules can be enforced by depending on user contexts (e.g. service provider publishes security context to limit a user's access to certain service). For example, the security context identifies that the end user does not have a permission to a service, MailDaemon. As a result, IA sends information to the end user by using other services (e.g. using GlobalSMS, instead). In fact, we have been motivated by the following user scenarios:

Scenario 1

A customer, who is a vegetarian, plans to have a dinner. He is searching for a restaurant. He prefers going to the restaurant located near his house. Moreover, he wants a traffic report

Introduction

while he is driving to the restaurant so that he can avoid traffic jam. As a good planner, he also wants to check what the weather is going to be like before he leaves from the restaurant.

Scenario2

A customer, who plans to travel to France, wants a weather report in Paris. As an English speaker, he would like all information translated to English and sent to his email address. Moreover, as he had not booked for a flight yet, he wants to book for the flight departed from London to Paris on 10 June 2008.

Scenario3

A salesman is traveling around the city. There are five clients he plans to meet in one day. It is a good idea to be notified about the distance to the next client office in his current location so that he could calculate for a fuel cost based on the distance coverage. At the end of the day, he wants to send information about the deals he made during the day to his head office and he wants to relax by watching a movie in a theater near his current location.

In each user scenario, the end user requests for service compositions in natural language (i.e. English). Moreover, the end user requires some service composition to be composed only when certain conditions (i.e. depend on current user location) are satisfied. In this thesis, we determine to improve SCE with IA so that the SCE could at least fulfill user demands in these three scenarios. An architecture proposed in this thesis is shown in Figure 3. It is worth mentioning that the components, that are filled with colour in Figure 3, have been invented/developed in this thesis. In fact, we have also modified service repository, goal ontology, input/output ontology and knowledge sink. These components will be discussed later in chapter 3.

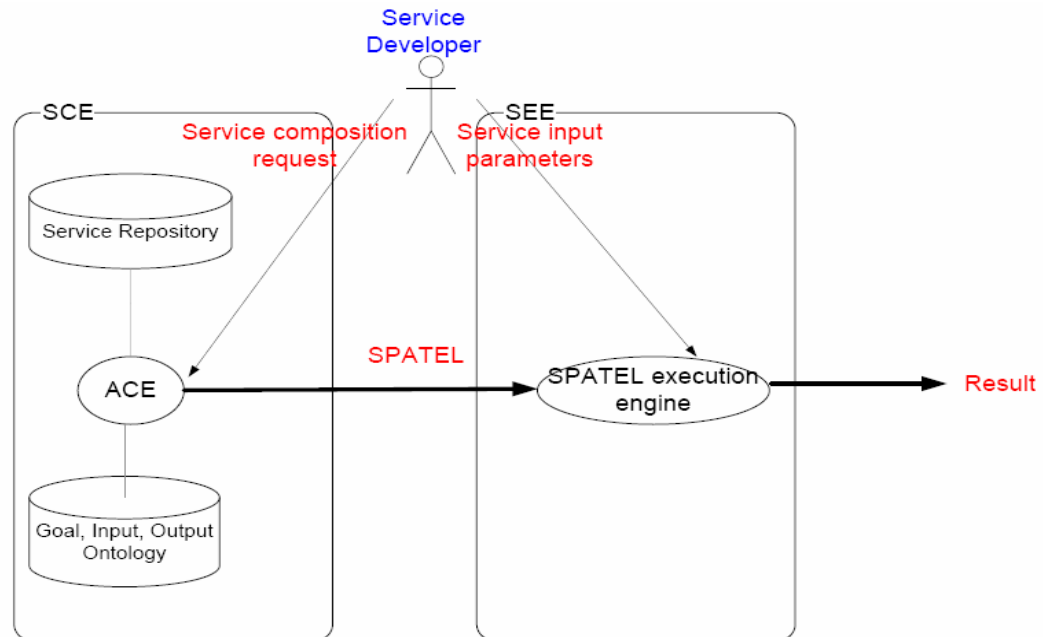


Figure 1: SCE and SEE architecture

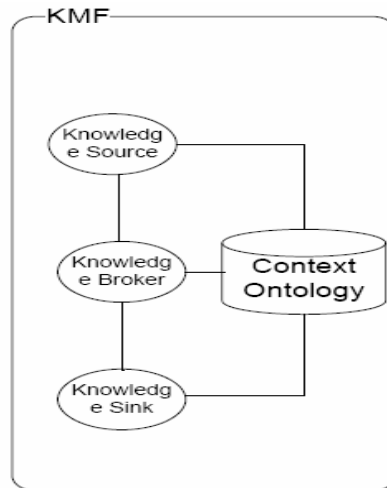


Figure 2: KMF architecture

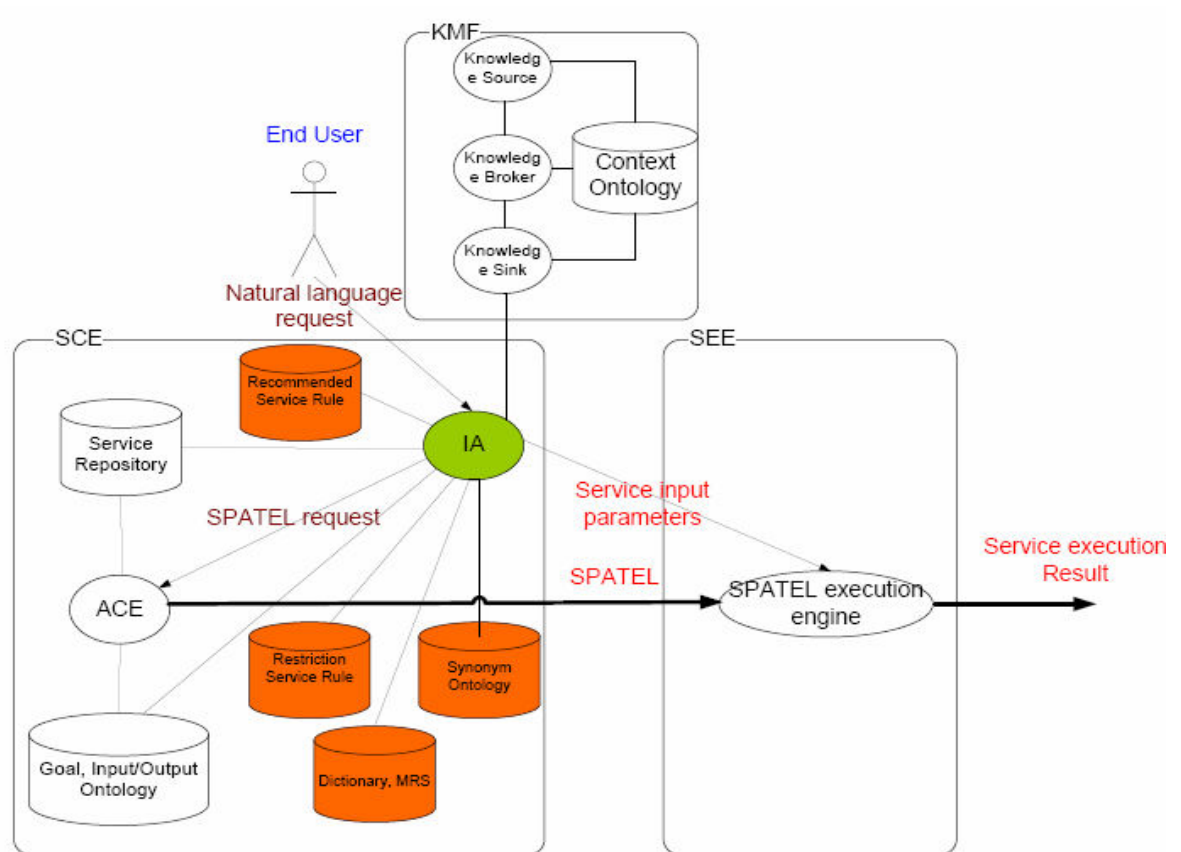


Figure 3: Our proposed architecture

1.2 Architecture Overview

As shown in Figure 3, IA provides an interface to the end user. Basically, the end user requests IA for service in natural language and the IA deduces, from the user request, 1) service compositions and 2) inputs of the service compositions. The most suitable service composition (selected by the user) together with its inputs can be passed to SPATEL execution engine (of SEE). SPATEL execution engine, then, invokes the service and delivers the result back to the end user. IA analyzes the natural language request and construct service composition, accordingly. Specifically, IA goes through two main phases, namely, discovery and composition. The first phase is to discover a list of consolidated services. The second phase is to compose and export the services to SPATEL (the process of export service composition to SPATEL is done via ACE).

IA has to conduct service discovery because ACE could not handle natural language request. In other words, ACE could not infer a list of service components from the input sentence. Similarly, IA has to conduct service composition because ACE could not perform service composition without knowledge about service components (e.g. what is the first service component to be invoked, what is the last service component to be invoked etc.). Therefore, IA has to supply ACE with these knowledge if ACE were to be used to generate service composition. However, we believe that the only way for IA to get these knowledge is to construct service composition first (based on semantic similarity of service component's input/output). This is because:

- We could not deduce the knowledge from a service component (to decide whether it is a start node or end node) without related its input/output with other service components. This means that we have to construct the service composition first
- We had to group service components that belongs to the same service composition together. The service components sent to ACE have to be components of one service composition. This means that we have to construct service compositions from the service components first

As a matter of fact, it is worth noting that the request that IA gives to ACE is more completed (i.e. have more information) than normal request that ACE takes from service developers. We called request from IA as SPATEL request. Normally, the request from service developers has at least four parameters (i.e. information). These four parameters are start nodes (start nodes are service components that take at least one input parameter from the user (and not from other service components)), end nodes (end nodes are service components that do not provide input to other service components), list of service components to be composed and list of service operations (service operation is a function that service component provides to the user). ACE conducts service composition based on these four parameters. The service compositions are exported into SPATEL that are ready to be executed by SPATEL execution engine.

However, as IA conducts service composition by itself (this process is necessary because IA could not deduce start nodes and end nodes unless service composition has been derived), IA directly conveys the service composition to ACE. Therefore, it is not necessary for ACE to conducts service composition. Instead, ACE immediately exports the service composition (obtained from IA) to SPATEL. Consequently, our approach does not only help facilitating the end user but also helps reducing ACE workloads (i.e. the work to be done by ACE).

After ACE receives SPATEL requests from IA, ACE will generate SPATEL files for those requests. Ideally, IA shall select one SPATEL file of a service composition that is most appropri-

Introduction

ate to the user request. Then, IA feeds that SPATEL file (which corresponds to the service composition) and input parameters (which correspond to service components of the service composition) obtained from user request to the SPATEL execution engine. SPATEL execution engine will invoke the service composition with the input parameters and generate the result. However, at the time of writing this thesis, SPATEL execution engine is not fully available. The process of execution service composition has to be done manually. That is, if we were to execute service composition, we would have to get SPATEL file from ACE and input parameters from IA. Then, we would have to manually feed the SPATEL and input parameters to SPATEL execution engine in order to obtain the result.

As shown in Figure 3, IA is a context-aware application. IA utilizes knowledge sink in KMF to obtain user contexts published by knowledge source. As a result, IA can propose services to the user or filters out inappropriate service based on a derivation of user contexts. In this thesis, KMF (e.g. knowledge source) has been available for us. However, we have developed a knowledge sink (by using KMF API) as a module of IA to subscribe for the user contexts published by the knowledge source.

As shown in Figure 3, there are six important types of files utilized by IA, namely, service repository, recommended service rule, restriction service rule, synonym ontology, ontologies that are necessary for service discovery and service composition (i.e. goal ontology, input/output ontology) and repositories that are necessary for NLP (i.e. dictionary file and **Minimal Recursive Semantic (MRS)** file). Examples of these files are shown in Figure 4 - Figure 11. Detailed descriptions on these files will be provided in chapter 3.

Ideally, service repository, goal ontology and input/output ontology could be shared between IA and ACE. However, in this thesis, we have chosen to develop the new service repository, goal ontology and input/output ontology instead of using the one that ACE uses. This is because the existing repository, goal ontology and input/output ontology are limited to some domains that do not cover our scenarios in section 1.1. Nevertheless, we have chosen to use the same format of service repository, goal ontology and input/output ontology so as to maintain compatibility between ACE and IA.

“Component”	“Operation”	“Goals”	“Inputs”	“Outputs”	“NF”
“GoogleTranslate”	“EnglishToNorwegian”	“TranslateToNorwegian”	“EnglishText”	“NorwegianText”	

Figure 4: Service Repository file

```
<owl:Class rdf:ID="GetGeoDistance">
<rdfs:subClassOf>
<owl:Class rdf:ID="GetDistance"/>
</rdfs:subClassOf>
</owl:Class>
```

Figure 5: An example of ontology class and its relation with another ontology class in goal ontology

```
<owl:Class rdf:ID="AirportInfo">
  <rdfs:subClassOf>
    <owl:Class rdf:ID="Text"/>
  </rdfs:subClassOf>
```

Introduction

```
</owl:Class>
```

Figure 6: An example of ontology class and its relation with another ontology class in input/output ontology

```
<owl:Class rdf:ID="Get">
  <rdfs:subClassOf rdf:resource="http://www.w3.org/2002/07/owl#Thing"/>
  <owl:equivalentClass>
    <owl:Class rdf:about="#Find"/>
  </owl:equivalentClass>
  <owl:equivalentClass rdf:resource="#Search"/>
</owl:Class>
```

Figure 7: An example of ontology class and its relation with another ontology class in synonym ontology

```
! context rule, invoke operation of service when the context occurs
!
! Context      Service      operation      input parameter
! -----
!"street@telin.nl" "CITYTrafficReport" "ReportTrafficCondition" "Paris"
```

Figure 8: An example of recommended service rule

```
! restriction rule, restrict an access to service when the context occurs
!
! Context      Service
! -----
!"A2.07@telin.nl" "MailDaemon,EmailDelivery"
```

Figure 9: An example of restriction service rule

```
! convert norwegian vocab to english vocab
!
! Norwegian    English
! -----
"finne"        "find"
```

Figure 10: An example of dictionary file

```
<ep cfrom='0' cto='4'><spread>addressee-rel</spread><label vid='6'>
<fvpair><rargname>ARG0</rargname><var vid='5' sort='x'>
<extrapair><path>WH</path><value>BOOL</value></extrapair>
<extrapair><path>ROLE</path><value>ROLE</value></extrapair>
<extrapair><path>PNG.NG.NUM</path><value>NUM</value></extrapair>
<extrapair><path>PNG.NG.GEN</path><value>M</value></extrapair>
<extra-
pair><path>PNG.PERS</path><value>SECPERS</value></extrapair></var></fvpair>
</ep>
```

Figure 11: An example of XML node in MRS

Service repository serves the same role as **Universal Description, Discovery and Integration (UDDI)** that is a registry for publishing and discovering Web Services and enables businesses to share information about Web services (However, service repository used in this thesis has

Introduction

smaller scale (i.e. number of services is smaller) than UDDI and do not limit to web services (i.e. telecommunication services has also been considered)). As shown in Figure 4, service repository comprises of six columns that are component, operation, goals, inputs, outputs and **Non-functional property (NF)**. Component is a service name. Operation is a function that a service provides to the user. Technically, it is a name of function call (e.g. as in traditional programming language) that the user of service invokes. Goal, input and output represent objectives, input parameters and a consequence of running the service, respectively.

Recommended service rule (or context rule) is a table that stores matchmaking between context and service candidates. For example, in Figure 8, it tells that when user is driving (user location is street@telin.nl), IA should invoke an operation, *ReportTrafficCondition*, of service, *CITY-TrafficReport*, with input parameter, *Paris*.

Restriction service rule is a table that stores matchmaking between context and services that are not suitable for the context. For example, in Figure 9, it tells that when user location is A2.07@telin.nl), IA should not select two services, *MailDaemon* and *EmailDelivery*.

The synonym ontology file keeps a relationship between words (that have similar meaning). For example, it tells that “find” and “search” have the same meaning. Moreover, it can identify that “food” and “restaurant” are closely related to each other. This file is important because it provides flexibility to the end user. In other words, end user can choose the words he prefers when he requests for a service. For example, he can request either “**find** restaurant information in Paris” or “**search** restaurant information in Paris”. These two requests are identical so long as the synonym (find and search) are defined in synonym ontology. In Figure 7, it indicates that “find”, “get” and “search” are synonym of each others. This is because they are defined as equivalence class in the ontology.

Goal ontology file keeps relationship between service goals. For example, in Figure 5, it indicates that the two goals, *GetGeoDistance* and *GetDistance* are similar. Moreover, *GetDistance* is a general goal while *GetGeoDistance* is a specific goal. This is because *GetGeoDistance* is a subclass of *GetDistance* in the ontology. Goal ontology is used by IA during a process of service discovery. Specifically, IA searches for services that have service goal (as defined in goal ontology) corresponding to user request.

Input/output ontology file keeps relationship between service inputs/outputs. For example, in Figure 6, it indicates that the two service inputs/outputs, *AirportInfo* and *Text* are similar. In particular, *AirportInfo* can be used as *Text* but not vice versa. This is because *AirportInfo* is a subclass of *Text* in the ontology. Input/output ontology is used by IA during a process of service composition. Specifically, two services can be connected if output of one service can be used as input of another service.

Dictionary file keeps a mapping between Norwegian word and English word. IA utilizes an application developed by linguistic experts to support full NLP functionality. We called this application as **NLP-tool**. NLP-tool analyzes user request and produces a semantic representation of sentence, that provides necessary information, for IA to conduct NLP. The compatibility issue is arisen because NLP-tool handles a request in Norwegian but the IA handles a request in English. For example, IA takes a user request: “find traffic in Paris and send them by email” but NLP tool takes a user request: “finne trafikk i Paris og sende dem pr. email”. We assume that for simple sentences (as the ones used for our demonstration purpose in the thesis),

English grammar is the same as Norwegian grammar. By this assumption, both IA and NLP-tool agree on a structure of sentence even though they are in different languages. In other words, a word in English sentence (e.g. find) always refers to another word (e.g. finne) in Norwegian as defined in the dictionary file. Hence, the word-to-word translation is sufficient to transform a result of NLP-tool into a form that IA understands. In Figure 10, it indicates that a Norwegian word, “finne” can be translated to English word, “find”.

MRS file is a file generated by NLP-tool. It indicates relationships between words in the user request (i.e. it is semantic representation file). It also indicates grammatical properties of words in the user request. This file is necessary when IA conducts NLP. In Figure 11, it indicates, for example, that a person who receives a request from the user is second-person (i.e. “you”).

1.3 IA components

In this section, we present inner parts of IA together with their workflows. The inner parts of IA are shown in Figure 12. The components, that are in green, have been developed in this thesis. The components, that are in orange, are available in the SPICE project. Although knowledge sink was available, we have modified it to suit with our work. We have modified knowledge sink so that it can process location context and security context. When the end user requests for service, the workflow of IA components is described as follows:

- 1 The end user requests a service via **Graphical User Interface (GUI)** of IA. The GUI will get the request and pass it to the parser. It is worth noting that the end user can enable/disable context-handling feature and MRS feature through the GUI.
- 2 When a natural language request from the end user is received, the parser will analyze the request. The parser will determine a grammatical structure of the sentence (e.g. which words are noun, which words are verb). This information (about grammatical structure) will be passed to information extraction module.
- 3 Information extraction module will extract all words (i.e. keywords) that are functioned as noun, verb and adjective (discussed in chapter 3). Then, it will pass these words to service discovery module.
- 4 The service discovery module will obtain services from service repository that correspond to keywords. Note that , if the context-handling feature is enabled, the services that are not appropriate for certain user contexts will not be selected (the services that are defined in restriction service rule will not be selected. The services that are restricted by knowledge sink will not be selected as well). The service discovery module will pass a list of consolidated services to input searching module and service composition module.
- 5 Keywords, obtained from information extraction module, and a list of services, obtained from service discovery module, will be passed to input searching module.
- 6 Input searching module will find keywords that can be inputs of certain services (by using input/output ontology). Input searching module will associate the keywords (that can be inputs of certain services) with services. For example, it associate “Paris” with service, “WeatherReport”. The pairs of services and input parameters (i.e. keywords) are passed to NLP-interface.

Introduction

- 7 The NLP-interface will verify that the pairs of services and input parameters agree with MRS (generated by third party application, NLP-tool). The pairs of services and input parameters that are validated by NLP-interface will be produced as outputs of IA. The pairs of services and input parameters that are not validated by NLP-interface will be removed. It is worth noting, however, that the end user can disable the verification feature via an interface of IA. In such a case, the pairs of services and input parameters will be produced as is.
- 8 When service composition module receives a list of consolidated services from service discovery module, it will produce service compositions according to semantic similarity of service input/output. The service compositions will be passed to sorter.
- 9 The sorter will sort a list of service compositions according to how well the service composition matches with user request and how well the service components of the service compositions are composed together. The sorted service compositions will be passed to NLP-interface.
- 10 The NLP-interface will re-arrange the service compositions again with a knowledge of MRS. The service composition that fails to agree with MRS will be degraded to the end of the list. The service composition that agrees with MRS will be maintained in the same position as they were sorted by sorter. The re-arranged service compositions will be passed to ACE-interface. It is worth noting, however, that the end user can disable the re-arranged feature via an interface of IA. In such a case, the service compositions produced from sorter will be passed to ACE-interface as is.
- 11 The ACE-interface will convert the service compositions into a format that is compatible with ACE. Then, the ACE-interface will pass these service compositions to ACE.
- 12 ACE will then construct the service compositions in SPATEL. These SPATEL files together with input parameters can be invoked by SPATEL execution engine.

The end user can enable context-handling feature of IA through its interface. When the location context is received, the workflow of IA components is described as follows:

- 1 When location context is received from knowledge source, knowledge sink read recommended service rule. Knowledge sink will construct service composition and generate input parameters as defined in the rule. The service composition generated from knowledge sink will be passed to ACE-interface.
- 2 The ACE-interface will convert the service compositions into a format that is compatible with ACE. Then, the ACE-interface will pass these service compositions to ACE.
- 3 ACE will then construct the service compositions in SPATEL. These SPATEL files together with input parameters can be invoked by SPATEL execution engine.

The end user can enable context-handling feature of IA through its interface. When the security context is received, the workflow of IA components is described as follows:

Introduction

When security context is received from knowledge source, knowledge sink will classify the security context. Security context can be of type authentication/authorization context or restriction context. If security context is authentication/authorization context for a certain service, knowledge sink will inform service discovery module not to select the service in response to the user request. If security context is restriction context for a certain service, knowledge sink will inform service discovery module that it is free to select the service in response to the user request.

We have discussed IA components and their workflows in general. For detailed descriptions, we refer to chapter 3.

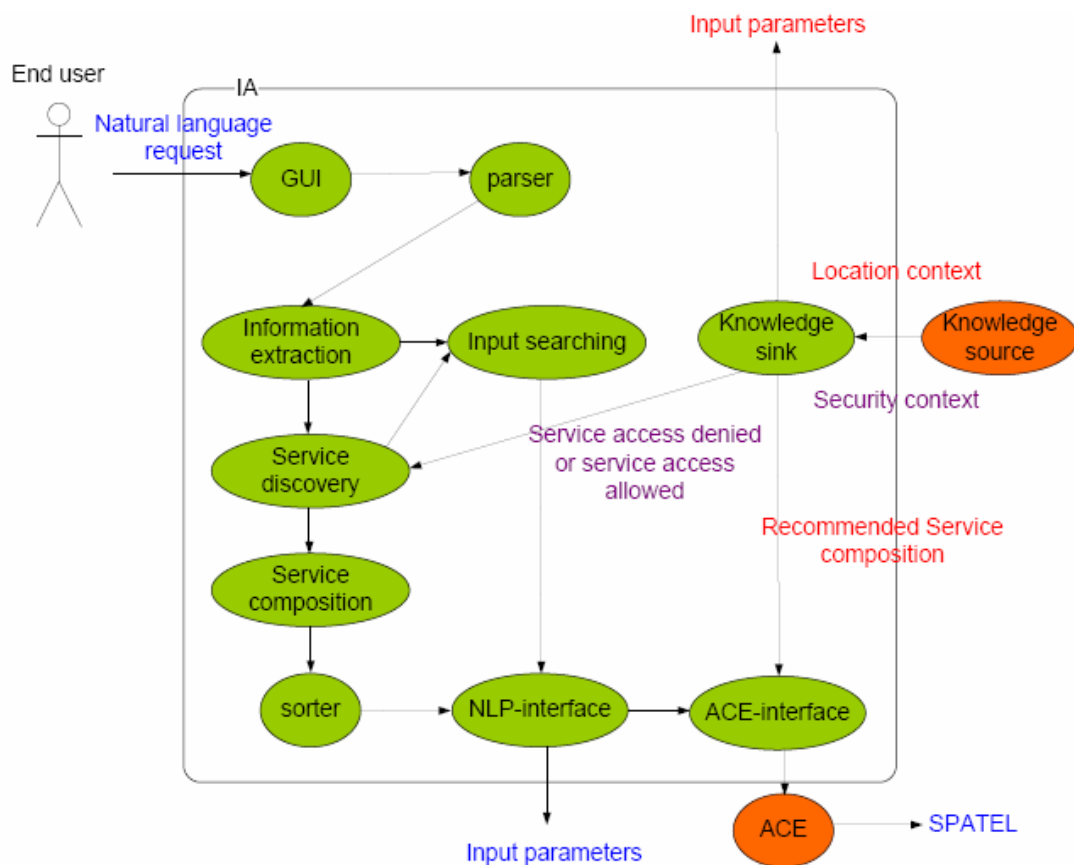


Figure 12: IA internal components

1.4 Thesis Scope and Limitations

This thesis focuses on investigating and demonstrating service discovery and service composition based on natural language processing and user context. The objective of this thesis is to demonstrate an idea to improve SCE with natural language processing and context handling features.

Our work consists of three main tasks. The first task is to research on related topics such as NLP and context handling. The second task is to develop a prototype of IA. The third task is to test and demonstrate the prototype with three user scenarios mentioned in section 1.1.

At the time of writing this thesis, the SPATEL execution engine is unfortunately not ready for automatic execution (we have to execute the SPATEL execution engine manually using 1) the SPATEL files and 2) the input parameter derived from user request) and the web service used for our demonstration purpose does not physically exist. Hence, we could not obtain the execution result from the SPATEL execution engine. Therefore, the scope of this thesis is limited to two frameworks, namely, SCE and KMF. Our work will not cover an execution of SPATEL by SEE. However, it is worth noting that the study on SPATEL and SPATEL execution engine are conducted in other research projects.

In this thesis, we will verify, among other things, that service compositions generated by IA is correct and appropriate to the user request (in natural language). We will also validate the context handling feature. Specifically, we will ensure that IA completely supports the three user scenarios proposed in section 1.1.

As for a limitation, we have developed IA on eclipse platform. The prototype of IA operates on PC. Therefore, the prototype of IA does not run on mobile devices. In future work, it would be a challenge to port IA to be able to run on mobile devices (considering that there are limited resources in mobile device and scalability issue could be arisen).

Due to time limitation, we limited our works on security topics. Although we demonstrated how user contexts can be used to restrict service accesses of the end user, we did not look into the details of a process to handle user login or user registration etc. This is because there has already existed a mechanism in SPICE project to support this functionality. Rather, we focused on the innovative topics such as natural language processing.

Moreover, we have assumed a type of services, that we used in this thesis. The services in this thesis are information services and application services. An example of information service is WeatherReport. An example of application service is GlobalSMS. However, it is worth nothing that, rather than focusing on services themselves, we focus on how these services can be composed together in response to the user request.

In conclusion, the thesis aims on demonstrating technical feasibilities and solutions and proposing a way to improve SCE so as to realize business opportunities.

1.5 Research Methodology

During a prototype development, an iterative software engineering approach [8] has been utilized systematically. Initially, the core functionalities of IA have been developed. The core functionalities include abilities for IA to provide simple interface to the end user, to subscribe

Introduction

to user contexts (by implementing context sink), to extract keywords from user request, to retrieve concepts (e.g. relationship between service and keywords) that are relevant to the keywords from ontology files and to discover a list of services. After these core functionalities had been implemented, additional features were added to the IA. These additional features are:

- NLP
- Service composition
- Service composition ranking
- Location-awareness
- Authentication-awareness

NLP has been given the highest priority. This is because we realized that NLP is a functionality that the current SCE lacks. It is the most challenge part of this thesis. After NLP had been implemented, we proceeded with an implementation of service composition. IA conducts service composition based on a result of NLP. Then, we implemented service composition ranking. Service composition ranking provides a functionality to rank service compositions based on how well they satisfy user request. With these features (NLP, service composition and service composition ranking), SCE can process a natural language request and generate service composition ranked according to its appropriateness to the request.

Two features (location-awareness and authentication-awareness) have been given lower priority than the other three features. This is because we realized that there had already been a mechanism supporting these features (i.e. KMF) available in the SPICE project. Therefore, we could utilize the existing technology (KMF) to integrate location-awareness and authentication-awareness to SCE rather than inventing a new approach.

As aforementioned, an iterative software engineering approach has been utilized. This is because the iterative software engineering provides a flexibility way in which we can add and remove features without suffering from degradation on product quality (i.e. IA). In other words, the low-priority features (e.g. authentication-awareness) can be implemented only when the high-priority features (e.g. NLP) have been completed. On the other hand, the high-priority features can be developed by applying a full software methodology process. As a result, the effectiveness and correctness of prototype (i.e. IA) can be ensured.

Ideally, the iterative software engineering process that we used can be described as follows:

- Initially, we conducted a research on related technologies and related works (e.g. NLP techniques, semantic web technologies). We selected the technologies that are most appropriate for our purpose (e.g. we prefer an NLP approach that does not put a restriction on user requests, we prefer an ontology language that can describe complexity attributes (relations) between ontology concepts). We implemented software prototype using the selected technologies. The first software prototype included the core features.
- After the software prototype had been developed, we experimented with the prototype. The prototype of the software was tested to verify its features (e.g. NLP).
- When test had been successful, the feature (e.g. ontology file is correct, NLP is suitable for a user request) was verified. We picked up another feature and the development

Introduction

loop was continued until all features were supported. By this way, the end result of the prototype was gradually improved [8].

- If the test were not been successful, we investigated and corrected the approach we used (e.g. we checked that the ontology were defined correctly), corrected the prototype (e.g. by debugging the source code, set up a breakpoint in the code) and proceeded until the test success. Only then, we added another feature to the prototype and processed further.

Once all features had been added, we tested the prototype with three user scenarios described in section 1.1. We have tested the prototype in two aspects: correctness and performance. The results of our testing are shown in chapter 5. Indeed, there could be some problems during the iterative software engineering process. For example, we may overestimate or underestimate a time to implement and test some features. This may make us unable to follow the timeline we have set up. In such a case, we may have to modify time schedule accordingly. In fact, we had modified time schedule three times as discussed in appendix B. It is worth noting, however, that eventually we have completed our works on time.

The division of our works done in this thesis (in percentage) is shown in Figure 13. As shown in Figure 13, we have concentrated on implementation and testing of the prototype (e.g. the overall implementation and testing account for 53 % of all our efforts). Especially, we have concentrated on implementation and testing of NLP features. Moreover, implementation and testing of fundamental features account for 18 % of all our efforts. This is because we have included designs and implementations of IA repositories such as goal ontology, input/output ontology, synonym ontology, service repository, recommended service rule and restriction service rule. The design and implementation of dictionary file has been included during a design of NLP feature.

Tasks	Efforts (in percentage)
Research on technical topics (NLP, semantic webs, KMF, SCE)	21 %
Demonstration of prototype	26 %
Design, implementation and testing (fundamental features)	18 %
Design, implementation and testing (NLP)	14 %
Design, implementation and testing (service composition and ranking)	7 %
Design, implementation and testing (location-awareness)	7 %
Design, implementation and testing (security-awareness)	7 %

Figure 13: Division of works

1.6 Thesis Outline

Chapter 1 provides an introduction to this thesis. It provides a background on this thesis, user scenarios that motivate the author, an overview of our proposed architecture, a scope and limitation of this thesis as well as a method that we used to conduct this thesis.

Chapter 2 is dedicated to provide all theoretical backgrounds and discussions on related technologies used in this thesis. It comprises of discussions on five related technologies. The first discussion is contributed to NLP technology. The first discussion is important for a design and implementation of NLP feature of IA. The second discussion contributes to semantic web technologies such as ontology concepts used through out the project. The third discussion elaborates on ACE and SPATEL. The fourth discussion provides backgrounds on context handling, specifically, the **KMF (Knowledge Management Framework)**, a framework used to handle user contexts in SPICE. The fifth discussion is devoted to provide some security background. The fifth discussion describes aspects that we need to consider when developing security-awareness IA.

Chapter 3 elaborates on IA features that are considered while developing the prototype and also detailed design. The features discussed in section 1.5 will be divided into sub-features (i.e. functionalities). A design decision for each sub-feature will be discussed in more detail. We will also provide a broaden view of IA design in chapter 3.

Chapter 4 sketches out implementation details. It provides a walk through the source code. More importantly, it also provides class diagrams (that describe how each module in IA interacts with each other) of IA.

The outcome of this thesis is presented and discussed in chapter 5. Demonstrations on three user scenarios mentioned in section 1.1 are illustrated. Moreover, correctness and performance of IA are verified. Finally, chapter 6 provides a conclusion of this thesis and a discussion about possibilities for future works.

Chapter 2

Related technologies

This chapter is dedicated to provide all theoretical backgrounds and discussions on related technologies used in this thesis. Section 2.1 is contributed to a discussion on NLP technology (the natural language discussed in section 2.1 is English). Section 2.1 is important for a design and implementation of NLP feature of IA.

Section 2.2 contributes to semantic web technologies such as ontology concepts used throughout the project (especially in the process of service discovery and service composition). Section 2.3 provides a discussion about SCE with a concentration on ACE. Section 2.4 provides backgrounds on context handling, specifically, the **KMF (Knowledge Management Framework)**, a framework used to handle user contexts in SPICE. Section 2.5 is devoted to provide some security background. Section 2.5 describes aspects that we need to consider when developing security-awareness IA.

2.1 Natural Language Processing (NLP)

NLP [4] is a subfield of artificial intelligence and computational linguistics. NLP involves two main operations that are 1) automated generation and 2) understanding of natural human languages. NLP is an effort to provide a smooth communication between human and machines (e.g. computers). NLP is used to transform information expressed in human language to a form that machine understands (i.e. *natural language understanding*). On the other hand, NLP is also used to transform information, that machine produced, to a form that human understands (i.e. *natural language generation*).

NLP covers broaden technical areas. Among others are *natural language understanding*, *natural language generation*, *information retrieval*, *speech recognition*, *lip-reading* and *speech synthesis*. We will only discuss about the techniques used to implement IA. These techniques are *natural language understanding*, *information retrieval* (i.e. IA does neither respond to the user in natural language nor handles speech request). *Information retrieval* is a technique that is used to extract a keyword from sentence. *Information retrieval* approximates a meaning of sentence from meanings of individual words. *Information retrieval* does not consider how an interaction of words form a new meaning (i.e. *information retrieval* ignores preposition such as “and”, “to”, “from” etc). *Natural language understanding* comprises of two main processes, 1) semantic analysis and 2) semantic representation. *Natural language understanding* is a technique that is used to extract a meaning of the sentence as a whole. *Natural language under-*

standing considers how interactions of words form a meaning of sentence. *Information retrieval* is easy to be implemented but it is less precise than *natural language understanding*. On the other hand, *natural language understanding* is difficult to be implemented but it is more accurate than *information retrieval*. In this thesis, initially, we have implemented the *information retrieval*. Later, we have enhanced our works with *natural language understanding* in order to obtain more accurate results. It is worth noting that before *information retrieval* or *natural language understanding* are used, we have to get a grammatical structure of sentence. This allows us to know which words we should extract (i.e. in *information retrieval*) and which words should be connected to form a meaning (i.e. in *natural language understanding*).

In section 2.1.1 - 2.1.3, the details of parsing, information retrieval, semantic analysis and semantic representation are elaborated. In section 2.1.5, we will discuss about minimal recursion semantic (MRS) which is a meta-level language for describing semantic structures. Indeed, MRS is a specialization of semantic representation approach discussed in section 2.1.4.

2.1.1 Parsing

A sentence needs to be assigned a grammatical structure before the meaning can be associated with it. This process of assigning a grammatical structure to the sentence is called parsing [4]. There are two basic approaches used in parsing [4]: a top down parsing and a bottom up parsing. A parsing can be viewed as a search process finding a parse tree to be matched with an input sentence. The parse tree can be found by gradually applying grammatical rules to each constituent in the sentence.

A top-down approach starts from generating a root of the parse tree by taking from the *sentence element* in grammatical rules (e.g. S in Figure 16). The grammatical rules have a form of **Context Free Grammar (CFG)** in which a non-terminal on the left hand side can be expanded into the elements on the right hand side [9]. CFG is appropriate for NLP because it allows us to present complex sentences by a parse tree. In other words, CFG is flexible enough to define complicated grammatical rules [9].

The sentence element is considered to be a root of the parse tree because it entails all the constituents in the sentence. An example of a grammatical rule for the sentence element is, for example, $S \rightarrow NP VP$ meaning that a sentence is composed of noun phrase and verb phrase. The noun phrase can be expanded into determinative and nominal noun and so on. An example of valid parse tree, for a grammatical rules defined (as CFG) in Figure 15, is shown in Figure 14 (NP is noun phrase, VP is verb phrase, Det is determinative, Nom is nominal noun).

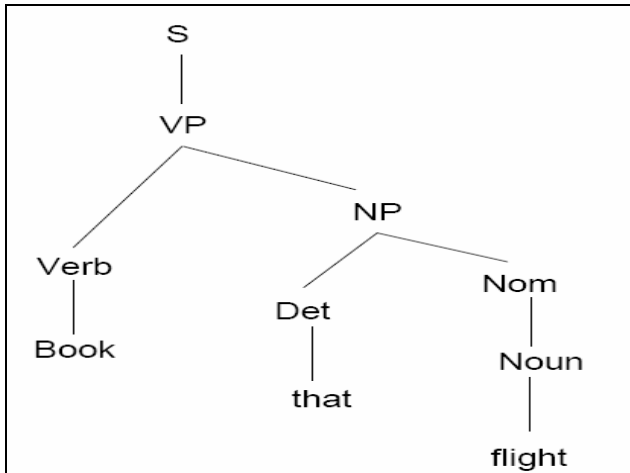


Figure 14: The correct parse tree for the sentence “Book that flight” according to the grammar in Figure 15 [4]

$S \rightarrow NP VP$	$Det \rightarrow that \mid this \mid a$
$S \rightarrow Aux NP VP$	$Noun \rightarrow book \mid flight \mid meal \mid money$
$S \rightarrow VP$	$Verb \rightarrow book \mid include \mid prefer$
$NP \rightarrow Det Nominal$	$Aux \rightarrow does$
$Nominal \rightarrow Noun$	
$Nominal \rightarrow Noun Nominal$	$Prep \rightarrow from \mid to \mid on$
$NP \rightarrow Proper-Noun$	$Proper-Noun \rightarrow Houston \mid TWA$
$VP \rightarrow Verb$	
$VP \rightarrow Verb NP$	$Nominal \rightarrow Nominal PP$

Figure 15: A miniature English grammar and lexicon [4]

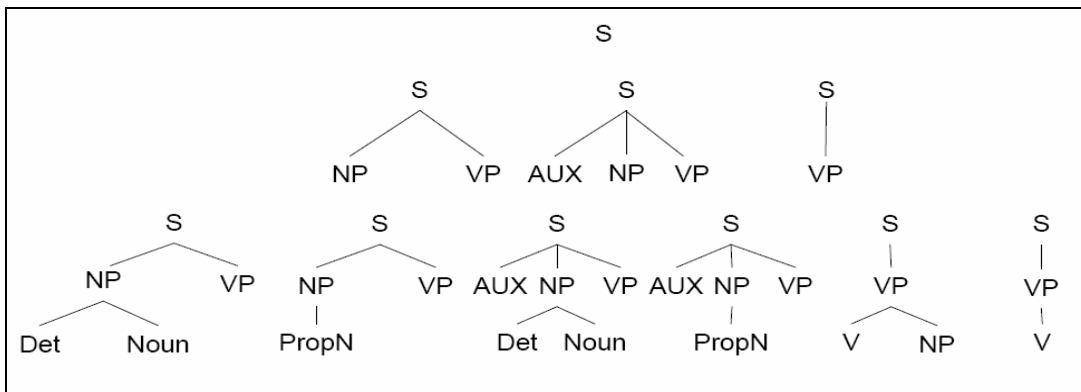


Figure 16: An example of parse trees generated by top-down approach

In top-down approach, grammatical rules are recursively applied to the non-terminal (e.g. NP, VP) until all terminal elements are derived (terminal elements are “Book”, “that”, “flight”). In top-down approach, several parse trees are generated as shown in Figure 16. Parse trees that fail to derive to a given sentence (e.g. “Book that flight”) such as the one in the right corner of Figure 16 will be removed. Eventually, only parse trees that can match with the given sentence will be generated.

Since a grammatical rule for one non-terminal element can be ambiguous (e.g. we have $S \rightarrow NP VP$, $S \rightarrow Aux NP VP$, $S \rightarrow VP$), we could have obtained multiple parse tree candidates. The leaf nodes of the candidates (i.e. the terminals) are compared against the input sentence in order to select the right tree. If there are two or more candidates matched with the input sentence, an ambiguity (i.e. syntactic ambiguity) will be arisen. We can remove the ambiguity by utilizing a statistical theory. For example, “book” is more likely to be a verb rather than a noun in certain context (e.g. if we are talking about a reservation process). Hence, one parse tree has high probability than the others to be correct. Sometimes, the ambiguity cannot be resolved and a parsing returns all possibilities parse trees together with its probability.

A disadvantage of top-down approach is that the effort might be wasted to generate trees that do not correspond to input sentence [4]. This is in contrast to the bottom—up approach in which a process starts from terminal elements (i.e. words in an input sentence). The non-terminals on the left hand side of grammatical rules will be searched to find the one matched with the terminal elements. The process will then recursively apply grammatical rules until the sentence element (i.e. S) is generated. The tree that does not result in the sentence element will be disregarded as it does not belong to a correct grammar (e.g. the one in the left cornet of Figure 17). If two or more correct trees are generated, the ambiguity will be arisen. The ambiguity can then be removed using the same technique as in top-down approach. An example of parse trees generated by bottom up approach is shown in Figure 17.

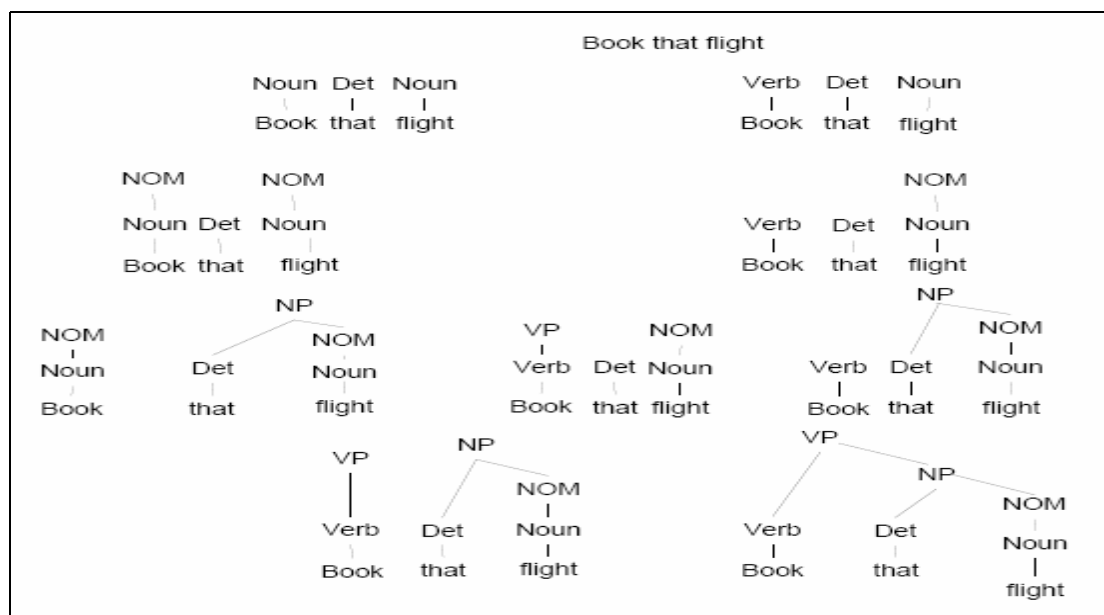


Figure 17: An example of parse tree generated by bottom-up approach

A disadvantage of bottom-up approach is that the effort might be wasted to generate trees that do not result in a correct grammar structure [4]. An optimal approach is to use top-down approach augmented with a bottom-up filtering. The optimal approach uses a top-down approach to generate a tree by applying the left-most depth first schema. That is, the rule is applied to the left-most terminal until it results in either a terminal matched with a word in the input sentence or an element that cannot match with the input and triggers a backtracking. Rather than waiting until the full parse tree is constructed, the bottom-up approach screens out all candidates that

fails to match with the input sentence as soon as it finds one. In fact, the library (will be discussed in chapter 3) that we used to parse user request utilizes the optimal approach (i.e. top-down approach with bottom-up filtering). This is because it provides a superior performance over traditional top-down and bottom-up approach (as discussed above).

Similar to top-down approach and bottom-up approach, the optimal approach has problem with ambiguity. When structural ambiguity arises, we may choose to produce all possible parse trees that are appropriated for the sentence. Unfortunately, the number of possible parse trees can be abundant. This makes the problem intractable for a complex grammar. To reduce the operation cost (i.e. the cost for processing all plausible parse trees), our design decision is to select the parse tree, that has highest probability, to operate on while ignoring the rest.

2.1.2 Information Retrieval

Information retrieval [4] is a technique that is used to extract keywords from a sentence. Information retrieval approximates a meaning of sentence from meanings of individual words (in fact, it approximates a meaning of sentence from meanings of “some” keywords in the sentence). Information retrieval does not consider how an interaction of words form a new meaning (i.e. information retrieval ignores preposition such as “and”, “to”, “from” etc).

Information Retrieval [4] is one of the most coarse-grained approaches used to extract key knowledge out of the text. It is based on a notion that the meaning of a sentence could be formed by a composition of the meaning of individual words in the sentence. The syntax or the word order is not considered to have any effects on the meaning of the full sentence.

Information Retrieval comprises of two processes. Firstly, we tag words (i.e. we indicate whether the word is noun, verb, adjective etc.) in the sentence. That is, we parse the sentence as described in section 2.1.1. Secondly, we retrieve keywords based on grammatical rules. In this thesis, we treated all nouns, verbs and adjectives in the sentence (i.e. user request) as keywords. This is because the words that has type noun, verb or adjective are more likely (than other words) to indicate service goals in our scenario. As we do not take an order of words in to account, sentences such as *Translate Weather information to English* and *Translate English to Weather information* are deemed to be the same. The keywords extracted using information retrieval technique are Weather and Translate which function as noun and verb in the sentence, respectively. An inability to distinguish the sentences with different orders is a drawback of information retrieval technique. We can improve the result of information retrieval technique by natural language understanding technique (discussed in section 2.1.4 and 2.1.3).

Information Retrieval technique is normally deployed by a search engine system to get keywords from user query. Basically, the search engine retrieves contents that websites publish. Then, the search engine selects websites that publish information containing keywords (the keywords are retrieved from the user query using information retrieval technique). The search engine may discover more than one website that provides content corresponding to the keywords in the user query. In this case, the search engine ranks the websites based on a degree in which they satisfy the user query. For example, if contents of the first website contain one keyword while contents of the second website contain two keywords, the second website will get higher rank than the first website.

In this thesis, we used information retrieval technique for service discovery. For service discovery, information retrieval technique is preferred over natural language understanding technique. This is due to its simplicity (information retrieval is simpler to implement than natural language understanding technique and its operation cost is even lower than the counterpart). Moreover, information retrieval technique is accurate enough for service discovery. It is worth noting that although information retrieval technique cannot distinguish two sentences such as *Translate Weather information to English* and *Translate English to Weather information*, a correct list of consolidated services are derived (e.g. *GoogleTranslate* and *WeatherReport*).

We used information retrieval technique to find a list of consolidated service as follows: Firstly, we tag words (i.e. we indicate whether the word is noun, verb, adjective etc.) in the sentence. That is, we parse the sentence as described in section 2.1.1. Secondly, we retrieve keywords based on grammatical rules. We treated all nouns, verbs and adjectives in the sentence (i.e. user request) as keywords. Then, we look into a service repository. We select services that have service goal corresponding to the keywords we extracted (that is, we select services that the service goal containing one or more keywords). If more than one service are selected, we will rank the services based on a degree in which their service goals satisfy the user query. For example, if the first service has goal corresponding to one keyword while the second service has goal corresponding to two keywords, the second service will get higher rank than the first service.

2.1.3 Semantic Analysis

Semantic analysis [4] is a process to derive and transform a meaning of sentence in to a form that machine understand. That is, the sentence will be represented by **First Order Predicate Calculus (FOPC)**. The semantic representation will be discussed in section 2.1.4. In this section, we discuss on how meaning is extracted from the sentence.

There are two well-known techniques of semantic analysis. These are **syntax-driven semantic analysis** and **semantic grammar**. The syntax-driven technique is used to attach semantic representation to each syntactical element in the grammar. The meaning of the sentence is formed by connotations of every word, their orders and their relations in the sentence. The syntax-driven semantic analysis does not take the context into account. In contrast, the meaning of the sentence in semantic grammar depends on a domain in which the sentence occurred (e.g. what is the topic that we are talking about). In both techniques, the syntactical element is generated by an appropriate parser before supplying to semantic analyzer as shown in Figure 18.

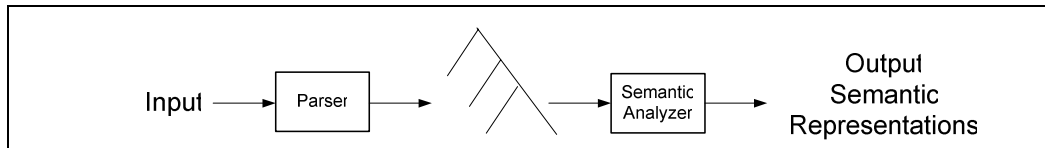


Figure 18: A simple pipeline approach to semantic analysis

For syntax-driven semantic analysis, a meaning is bound to each grammatical element. The rules for semantic attachment are enforced while the syntactical structure is constructed. Different rules are used to impose semantic attachments based on types of grammar. For example, there are different rules for Noun Phrases, Verb Phrases and Preposition phrases. The semantic of a sentence as a whole is constructed by combining each semantic component in the sentence.

A meaning of one element can be correlated with another element. In such a case, a lambda notation (i.e. $\lambda xP(x)$) is used to convey this information. The lambda notation is a variable

binding approach, in which a variable, x , in one semantic representation can be bound to a semantic of an object, $P(x)$, (may be in another semantic representation) that forms a relationship with the representation.

One shortcoming of syntax-driven semantic analysis is that the semantic attachment rule is too generic. It could produce many vacuous meaning representations. For example, one syntactical element can be bound to several meanings (this is called semantic ambiguity as opposed to syntactic ambiguity in section 2.1.1). This makes a sentence has more than one meaning. To overcome this, a semantic grammar is designed to be specific to a domain. Hence, the number of interpretations for the given sentence is bound to the size of the domain. Moreover, a syntax-driven semantic analysis requires complicate structure to start with. Sometimes, it requires a lambda notation variable to represent unforeseen semantic object that could not be seen in the current grammatical level (e.g. preposition phrases) and elevate that variable to the higher grammatical level (e.g. verb phrases) by nested inside another lambda notation. Since the semantic grammar operates in one specific domain, it has flat structure (i.e. no unforeseen semantic object as it can make an implicit assumption on the domain) and does not require coping with the lambda notation complexity.

The semantic grammar has also an advantage in searching for candidates of a pronoun in dialogue system (in fact, the semantic grammar has been used for the work of **Natural Language Processing (NLP)** presented in [3]). Providing a domain, one can search for a subject related to the topic with a high possibility that it is a good substitution for the pronoun. However, one primary drawback of a semantic grammar is its reusability. Since the grammar and semantic analysis is designed to be specific to a single domain, the work could not be deployed in other systems. It also agonizes from a semantic over-generation. The semantic grammar is based on a general sense assumption on the domain. Therefore, a different use of the word meaning (e.g. a word “book” can be used for two meanings. One is for “manuscript” and another one is for “reserve”) might not be processed correctly. Despite these drawbacks, NLP-tool that we used to generate a meaning of sentence (i.e. user request) is based on semantic grammar approach. This is because we concern on ambiguity of sentence if syntax-driven semantic analysis were to be used. Moreover, as we used information retrieval rather than semantic grammar in a process of service discovery (we used semantic grammar only to verify service compositions), we would not suffer from the limitation of semantic grammar (that the service which has different meaning of goal may not be selected). This is because the information retrieval selected services that have goal corresponding to all meanings of keywords.

2.1.4 Semantic Representation

In section 2.1.2, we represent the information retrieval technique. The techniques do not consider grammatical structure of sentence (e.g. user request). In some situations (e.g. in some situations, natural language understanding is not necessary. For example, a search engine only uses information retrieval technique to find appropriate websites), the machine needs to fully understand the requests (by considering grammatical structure) in order to fulfill user needs.

Therefore, the semantic analysis in section 2.1.3 is used to extract meaning from a sentence. The meaning extracted from the sentence is presented in a form that machine understands. This is based on a fact that human language structure can be transformed into a function argument structure that the machine can be absorbed [4]. A process to represent a meaning in a form that machine is understand is called semantic representation. The semantic representation is re-

quired for the machine to correctly handle conditional queries and correctly grasp user contexts.

A First Order Predicate Calculus (FOPC) [9] is utilized to convert a sentence (e.g. user request) into logical formulas that the machine can understand. The machine can verify a validity of the logical formulas (e.g. machine verify whether the information presented by formula is correct) with knowledge (that are also represented by logical formulas) stored in the machine's database. The sentence is represented by a predicate or a function which serves to represent a relation between different objects (i.e. words) in the domain (i.e. sentence). The grammatical structure of the sentence is recognized in order to transform the free form language into FOPC.

The objects in FOPC are considered as *Terms* and are represented by *constants*, *functions* or *variables*. A constant represents an identifiable object in the sentence such as a restaurant name. Function represents an object that forms a connection with other objects such as a genitive, a location of the restaurant. A variable represents an anonymous object or all objects in the domain. Relationships between *Terms* are represented in a form of *predicate*. To clarify on this point, a sentence such as “Maharani serves vegetarian food” [4] is represented by $Serves(Maharani, VegetarianFood)$. In this example, $Serves$ is a predicate. $Maharani$ and $VegetarianFood$ are constant. Thus, predicate connects two *Terms*, namely, $Maharani$ and $VegetarianFood$.

Moreover, the composite of relationships (i.e. a connection between two predicates) can be established using logical operators (e.g. and, or, not). Quantifier operators such as \forall (for all), and \exists (for some), are also employed to give a meaning to variables by binding them to their scope (i.e. for some or for all). For example, $\exists x Restaurant(x)$ indicates one *Term*, x , of type variable. $\exists x Restaurant(x)$ can be read as “for some restaurant, namely, x ”.

A set of FOPC equations could be used to infer an implication in the sentence (e.g. user request). For example, we could grasp from a context of the sentence that all vegetarian restaurant serves vegetarian food. If Rudys were a vegetarian restaurant, it would have also served vegetarian food. The inference can be obtained using one of two techniques, namely, **forward chaining** and **backward chaining**. In forward chaining, if one would like to infer a goal, one has to start from an existing rule in the database and work out a way to the final goal. Along the path, irrelevant inference may be established and saved into the database. As for backward chaining, if one would like to infer for a goal, one has to assume initially that the goal is valid and prove for the precedence that leads to the goal. The same process is repeatedly applied to the precedence and its precedence until one can derive the explicit statement from the sentence (e.g. user request). For example, suppose that we have two predicates: $P(x,y)$ and $P(y,z)$. The first predicate indicates that x is a parent of y . The second predicate indicates that y is a parent of z . In order to derive a third predicate, $P(x,z)$ indicating x is a parent of z . In a forward chaining approach, we start from the two existing predicates, $P(x,y)$ and $P(y,z)$. Then, we establish the third predicates (i.e. $P(x,z)$) by knowing that x is a parent of z if and only if x is a parent of y and y is a parent of z (Apparently, if y is z , we will not need to establish the third predicates as the condition ($P(x,z)$) is derived from the beginning). In a backward chaining approach, we start from assuming that the predicate $P(x,z)$ is valid. Then, we prove that two predicates $P(x,y)$ and $P(y,z)$ are, in fact, valid. In other words, if $P(x,y)$ and $P(y,z)$ are in database (i.e. in our knowledge), we prove the third predicate $P(x,z)$. Otherwise, we recursively repeat the process by assuming, for example, that $P(y,z)$ is valid and prove the two predicates such as $P(y,k)$ and

Related technologies

$P(k,z)$ and so on. Eventually, we either prove that the predicate is valid or invalid as we exhausted a search in database.

In essence, the FOPC can be deployed to express certain concepts from the input sentence such as categories, events, time, aspect and beliefs. A technique called **reification** [4] is used to introduce new artificial relation in FOPC to avoid violation on the condition that the argument of predicate in FOPC has to be object (i.e. *Term*) but not predicate and, therefore, increase flexibility of the FOPC itself.

A category concept [4] expresses a classification of objects. The categories are established using artificial “is-a” relationship [4] and these categories can be expressed in other categories to form a hierarchy of concepts. An expression of category concept is, for example, $ISA(Maharani, VegetarianRestaurant)$ [4]. This illustrates that an object (Maharani) belongs to (is-a) a category (VegetarianRestaurant). Another example is $AKO(VegetarianRestaurant, Restaurant)$. This indicates that one category (VegetarianRestaurant) belongs to (i.e. is a kind of) another category (Restaurant). Thus, $AKO(VegetarianRestaurant, Restaurant)$ forms a hierarchy between categories.

An event concept [4] expresses types of incident. The reification technique can be used again to support a type of logical inference. An expression of event concept is, for example,

$$\exists w ISA(w, Eating) \wedge Eater(w, Speaker) \wedge Eaten(w, TurkeySandwich) [4]$$

This indicates that there is an eating event where the Speaker eats (doing the eating) turkey sandwich (which is being eaten). w indicates this event (i.e. the first formula indicates that there is some eating events).

A time concept [4] is described using **temporal logic** [9] (e.g. propositional temporal logic). This concept distinguishes two sentences describing the same event but happening at difference times. The **reference point** [4] might be used to justify that one event in the sentence precedes another event in the sentence if the first event occurs before the reference point while the second event occurs after the reference point. An expression of time concept is, for example,

$$\exists i, e, w, t ISA(w, Arriving) \wedge Arriver(w, Speaker) \wedge Destination(w, NewYork) \\ IntervalOf(w, i) \wedge EndPoint(i, e) \wedge Precedes(e, Now) [4]$$

$$\exists i, e, w, t ISA(w, Arriving) \wedge Arriver(w, Speaker) \wedge Destination(w, NewYork) \\ IntervalOf(w, i) \wedge MemberOf(i, Now) [4]$$

$$\exists i, e, w, t ISA(w, Arriving) \wedge Arriver(w, Speaker) \wedge Destination(w, NewYork) \\ IntervalOf(w, i) \wedge EndPoint(i, e) \wedge Precedes(Now, e) [4]$$

The first example indicates a sentence “I arrived in New York”. The second example indicates a sentence “I am arriving in New York”. The third example indicates a sentence “I will arrive in New York”. It is worth noting that a reference point (Now) has been used to indicate event time (e.g. an event that occurs after “Now” indicates a past).

Related technologies

An aspect concept [4] describes characteristics of an event whether it is *statives*, *activities*, *accomplishments* or *achievements events*. A *statives* expression is used to represent a status of the subject (e.g. speaker, user). For example, *statives* expression is used to represent a sentence “I know my departure gate”. The sentence indicates that a speaker is aware of his information. *Activity* expression is used to represent events (undertaken by the subject) that have no particular end point. For example, *activity* expression is used to represent a sentence “John is flying”. The sentence does not give information about an end point of the event. An *accomplishment* expression is used to represent events (undertaken by the subject) that have a natural end point and result in a particular state. For example, *accomplishment* expression is used to represent a sentence “Sally booked her flight”. The sentence indicates that the event has been finished at a particular state (of the event). An *achievement* expression is used to represent events (undertaken by the subject) that have a natural end point but the duration of the events is unknown. For example, *achievement* expression is used to represent a sentence “She found her gate”. The sentence does not indicate how long it takes for the subject to accomplish the event.

A belief concept [4] expresses a hypothetical world of the owner of the sentence. The FOPC needs to be augmented with an additional operator (e.g. Believe operator) to emphasize that the logical formula is valid only in the hypothetical world but not the real world. An expression of belief concept is, for example,

$\text{Believes}(\text{Speaker}, \exists v \text{ISA}(v, \text{Eating}) \wedge \text{Eater}(v, \text{Mary}) \wedge \text{Eaten}(v, \text{BritishFood}))$ [4]

The believe operator indicates that the expression inside the operator is realized by the subject (i.e. speaker). It may not indicate a real fact (e.g. “I believe that the world is flat”, “the world is flat” is valid in the imagination of the speaker but it is not the real fact).

It is worth to mention that there are other systems used to present the semantic of the sentence. Among others are **semantic networks** [4] and **frames** [4]. Semantic networks use graph style to present the semantic. The graph composes of nodes (i.e. semantic words) which are linked together by name reference to construct the semantic sentence. Frames use feature structure style [4] (i.e. a two dimension matrix keeping pair of attribute name and value) to present the semantic. However, the two systems can be transformed to the FOPC. The FOPC style for semantic representation is shown in Figure 19:

I want you to send me a SMS what the weather is going to be in Cuba in the first two weeks of April 2007

$\text{Weather}(\text{Cuba}(\text{weeks}(\text{first}(\text{two})), \text{april}(2007)))$, $\text{Send}(\text{Weather}, \text{SMS}(\text{me}))$

Figure 19: The semantic representation using FOPC

In section 2.1.3 and 2.1.4, we have discussed about natural language understanding technique. Natural language understanding technique extracts full meaning of sentence (such as user request). Compare to information retrieval technique, natural language understanding is more powerful. However, natural language understanding is more difficult (than information retrieval technique) to implement.

In this thesis, we have made two decisions regarding NLP techniques. Firstly, we have selected an NLP approach for service discovery. Secondly, we have selected an NLP approach for service composition. We have selected information retrieval technique for service discovery. This

is because, despite its simplicity, information retrieval is powerful enough to perform service discovery. Information retrieval may result in too many services got selected as a result of keyword-matching (e.g. a service that service goal contains “book” get selected despite the real meaning of “book” in the user request). However, information retrieval provides a safe solution (that is the result will always include the service that the user wants (with some other services)). One concern of information retrieval is that user may use other words rather than keywords in service goal to convey the same meaning. For example, user may request “reserve that flight” rather than “book that flight”. As the keyword “book” is not used, the service that has goal “BookFlight” may not be selected. We have solved this problem by using ontology concept discussed in section 2.2 (i.e. synonym ontology) to realize that “reserve” refer to the keyword “book”.

In contrast, we have selected natural language understanding approach for service composition. This is because information retrieval gives us no details about how services are composed together. It also gives us no details about input of services. For example, if a request is “send weather information in Paris to my mobile phone” and information retrieval is used for service composition, we will not know that the service WeatherReport should be invoked before the service SendSms. Moreover, we will not know that Paris is an input of the service WeatherReport. This is because information retrieval does not tell us about an interaction of events (or words) in the sentence. Although we can create service composition based on ontology concept (input/output ontology is used to provide information about service input/output. One service can be composed with another service if its input (or output) is compatible with another service’s output (or input)), the number of service compositions created based on this approach are too much (all possible of compositions are created). Similarly, although we can associate input instances (a word in the user request) based on ontology concept (a word is associated to a service if it is an instance or a synonym of an instance of the service input as defined in ontology), ambiguity could be arisen. For example, if a user request is “Book me a flight from London to Paris” and one of services in the service composition generated by this request is ReserveFlight (that takes two city names as inputs (one city name is departure city and another city name is arrival city)), we will not know which cities should be departure city and which cities should be arrival city without natural language understanding. Hence, natural language understanding has to be used to enhance a correctness of our results.

It is worth noting that the IA itself does not contain natural language understanding capability. Instead, it uses a natural language understanding capability of NLP-tool (that has been developed by linguistic experts). The NLP-tool is used to analyze the sentence. Then, the result (in FOPC described using MRS) is given to IA in a form of XML. IA then uses this result to enhance a correctness of service compositions and service input matching.

In section 2.1.4, we have discussed about FOPC. Several forms of FOPC have been presented (e.g. we have presented FOPC that has been extended with temporal logic etc). In this thesis, a normal form (as in Figure 19) of FOPC (generated by NLP-tool) is sufficient for our demonstration purpose. This is because normal form of FOPC is powerful enough to capture a relation of events (i.e. words) in a simple sentence (as in our scenarios). However, it is worth noting that if a complicate sentence were to be used, a complex form of FOPC would have to be deployed (in fact, NLP-tool that we used support this through MRS). For example, if a user request is “book that flight now”, FOPC with temporal logic will have to be used.

2.1.5 Minimal Recursion Semantic (MRS)

Minimal Recursion Semantic (MRS) [38] is a semantic framework (that is used for capturing semantic representation of sentence) designed with the following principles:

- Expressive adequacy
- Grammatical compatibility
- Computational tractability
- Underspecifiability

Expressive adequacy indicates an ability to express linguistic meanings correctly. Grammatical compatibility provides ability for a framework to tie with already existing grammars such as **Head-Driven Phrase Structure Grammar (HPSG)** [39]. HPSG is a framework that is used to capture grammatical structure of sentence as described in section 2.1.1. Computational tractability is an ability to capture a semantic representation of sentence in a reasonable amount of time. Underspecifiability is “an ability to leave the specifics of partial representation open while still retaining the validity of the underspecified expression” [38]. In other words, it provides an ability to represent semantic ambiguity of complex sentence while, at the same time, does not undermine a semantic representation of parts of the sentence that are unambiguous.

MRS is designed to solve three problems of semantic analysis that is based on a deep processing of text [38]. Firstly, the deep processing approach delves deeply into the underlying linguistic structure. This results in a vast search space. To solve this, the shallow processing approach (that requires lower overhead) [38] can be used first to generate an underspecified MRS representation of sentence (i.e. leaving some parts of sentence to be ambiguous). The MRS result can then be used to limit a vast search space of deep processing approach. Secondly, the deep processing approach fails to parse data that is considered ungrammatical due to insufficient lexical information. MRS solves this problem by reducing data sparsity [41] (data sparsity is a cause of insufficient lexical information). Thirdly, deep processing approach requires large amount of speed to analyze a sentence. As MRS reduces the vast search space, the speed is increased.

The basis structural units in MRS are called **elementary predications (EP)** [38]. EP indicates a single semantic relation with its required arguments. An example of EP is $\text{love}(x,y)$. EPs are not allowed to be embedded in one another. Therefore, syntactically flat structures (rather than deep or recursive structures) are maintained. There are several advantages of using syntactically flat structures rather than syntactically recursive structures. Firstly, application based on syntactically flat structures does not have overhead to process two representations which share the same meaning (as these representations are not allowed). For example, in syntactically recursive structure, we may have two representations for “a wooden box that is tiny” [38] and “a box that is tiny and wooden” [38]. The two representations are (note that lambda notation discussed in section 2.1.3 is used):

- $\lambda x(\text{tiny}(x) \wedge (\text{wooden}(x) \wedge \text{box}(x)))$ for “a wooden box that is tiny”
- $\lambda x((\text{tiny}(x) \wedge (\text{wooden}(x)) \wedge \text{box}(x))$ for “a box that is tiny and wooden”

Despite the fact that these representations share the same meaning, syntactically recursive structure approach generates two different representations for these sentences. In essence, the

application, that is based on the recursive structure approach, has to support these two representations. On the other hand, the application, that is based on the flat structure approach, has to support only one representation of these two sentences.

The reason that the flat structure approach does not embed syntactic information in semantic structure allows us to map a meaning between different languages. For example, if we would like to map a representation of “white English horse” [38] from English to German equivalence, “Schimmel” [38]. This could lead to a mapping problem caused from ambiguous bracketing [38]. If we represent the meaning of this sentence by using flat structure approach we get:

$\lambda x(\text{white}(x) \wedge \text{English}(x) \wedge \text{horse}(x))$ in English

$\lambda x(\text{Schimmel}(x))$ in German

Consequently, we can infer that the two representations are equivalence (because the two arguments, “white English horse” in English and “Schimmel” in German means the same thing). Apparently, there is no bracketing ambiguity problem. This is because the syntactic information (presented by bracket) is not embedded in the semantic representation.

Indeed, this is especially the case in this thesis as NLP-tool supports a user request in Norwegian (i.e. get a semantic representation of the Norwegian sentence) while IA supports the user request in English (i.e. need to map the semantic representation in Norwegian to English).

However, as the name implied, MRS still needs a minimal level of representing recursive structure in order to correctly interpret scopal structures (e.g. “every”, “some”) [38]. For example, if we have semantic representation:

$\lambda x(\text{every}(x) \wedge \text{horse}(x) \wedge \text{old}(x) \wedge \text{white}(x))$

This semantic representation can be a representation of “every horse is old and white”, “every white horse is old” or “every old horse is white” [38]. To remove this ambiguity, MRS allows a representation of embedded, recursive structure. However, if we do not have enough information to remove the ambiguity (if the sentence is syntactically ambiguous), MRS allows us to represent the ambiguity information with underspecifiability property [38].

Another way of viewing a semantic representation in MRS is to illustrate it with tree structure. For example, a semantic representation of “every big white horse sleep” [38] can be represented by recursive structure approach as shown in Figure 20. If MRS is used, the tree structure of depth n will be merged into a tree of depth 1 (indicating a flat structure). This is shown in Figure 21. It is worth nothing that “;” in Figure 21 indicates \wedge (i.e. “and” operation).

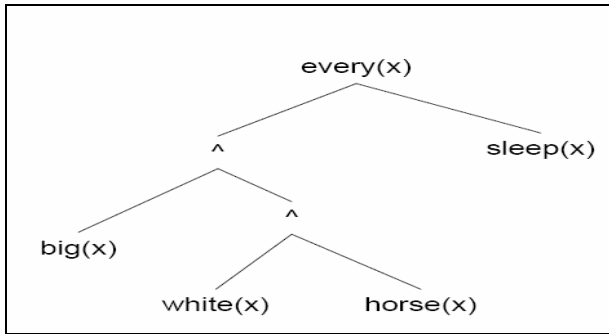


Figure 20: A semantic representation of “every big white horse sleep” based on recursive structure approach

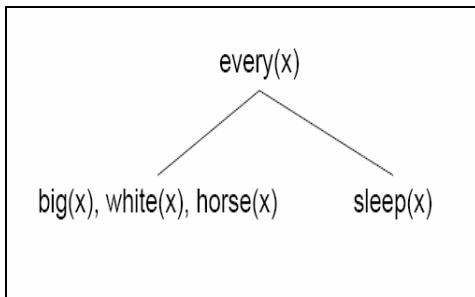


Figure 21: An abstract view on a semantic representation of “every big white horse sleep” based on MRS

To be more precise, each node in MRS tree are presented by a so called **handles** [38]. Handles identify an EP with a potential scopal argument position. For example, a MRS tree for “every big white horse sleep” is shown in Figure 22. Handle allows us to keep structures flexible enough for accommodating scope while still retaining a minimal recursive structure. Handle also allows us to present an ambiguity in the sentence. For example, a sentence such as “Every dog chases some white cat” [38] can be represented as shown in Figure 23. As shown in Figure 23, handle is not permanently tied with any EP (i.e. underspecified). Therefore, the underspecified handle can be bound with any EPs we selected. For example, h8 can be bound to h5 and h9 to h4 or h8 to h4 and h9 to h1. In the first case, it means that some of the cats (and only these cats) are chased by every dog. In the second case, it means that every dog chases some of the cats (every dog may not chase the same some cats).

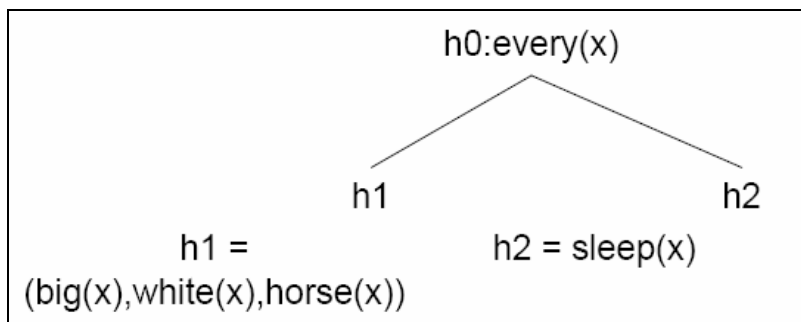


Figure 22: A semantic representation of “every big white horse sleep” based on MRS

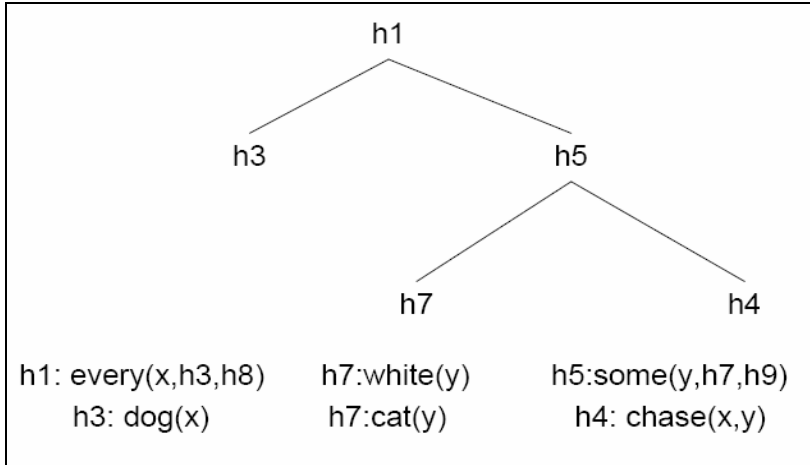


Figure 23: A semantic representation of “every dog chase some white cat” based on MRS

MRS is powerful enough to handle dislocated phrases [38]. For example, MRS can be used to represent the two sentences “Mary mentioned **the claim that John is intelligent** yesterday” [38] and “Mary mentioned **the claim** yesterday **that John is intelligent**” [38]. The first sentence emphasizes that the claim was made yesterday. The second sentence emphasizes the claim, “John is intelligent”, itself. Conventional semantic representation approach cannot be used to represent a full meaning of the second sentence. This is because it fails to capture a meaning out of a complex sentence. Specifically, it fails to map the modifier, “that John is intelligent”, with the modified element, “the claim” (the modifier and the modified element are separated by the word “yesterday”). MRS, on the other hand, can be used to represent both sentences by using handlers. Specifically, the two sentences can be represented as: h1: claim(x,h2), h2: John(x), h2: intelligent(x). In other words, despite what the sentences are emphasized on, MRS can grasp the core meaning of the sentences by intersecting the components, “claim”, “John” and “intelligent” using identical handles for all entities in the intersection. The meaning for both sentences is therefore:

- a claim has been made \wedge
- someone has been claimed to be intelligent \wedge
- this someone is John

In this section, we have illustrated how MRS is used for semantic representation of a sentence. In fact, in this thesis, MRS is used for semantic representation of user request. Several advantages of MRS have been discussed in this section. In particular, we have benefited from MRS as it provides capabilities to express semantic relations of user request adequately. For example, it can provide a semantic representation for a complex sentence such as the extraposed relative clauses [40]. It also provides capabilities to support an ambiguous sentence via underspecificity. A semantic analysis that is based on MRS representation can be done within a reasonable time. This is important for a real-time application such as IA. Lastly, as MRS is used, our application can be extended to support complex user request with ease.

2.1.6 Related works in service composition based on NLP

Our work is based on an approach presented in [27]. An analysis of free-form request is based on an examination of related keywords and their corresponding ontologies. It is also based on

synonym ontology (i.e. WordNet) to provide flexibility in operating on user request. However, [27] has more restriction than our work as it put some assumptions on the grammatical structure of the request. For example, it assumes that goal in user request must correspond to (i.e. occur after) keywords “maximize” or “minimize”. Keywords are classified into class, task, attribute and unrecognized words. Class keywords must occur after indefinite article. Task keywords must occur after preposition, “to”. Attribute keywords must have attribute name and attribute values associated with it. A word that fails to match with one of these types is an unrecognized word. A request that composes only of unrecognized word cannot be processed.

The process to extract and classify keywords is called Knowledge Fusion [27]. In contrast, our work does not take an order of the word in to account. It links keywords that are corresponding to service goals (and, hence, service name) by using ontology concepts. Thus, it is not bound by the grammatical rules or even a user choice of language (e.g. English or Norwegian). Our work is also more flexible than [28] which is based on a build of predicate logic from taxonomy features of vocabularies while assuming some patterns on the user request. However, our approach could provide too many service composition candidates depending on a sophistication of user request and associated ontologies. In this case, one requires utilizing a natural language processing tool (e.g. [29], [30]) from a linguistic expert to rank the service composition candidates properly (the most appropriate candidate comes first). The necessity for the natural language tool is also mentioned in [27].

2.2 Semantic Web Technologies

Current web technology (Web 2.0) aims to enhance creativity, information sharing and collaboration among users. Examples of the Web 2.0 technologies are social-networking sites, wiki, blogs and folksonomies. Web 2.0 technologies use syntactical information of HTML document to retrieve and share information among web clients.

Web 3.0 technologies are an effort to move beyond the current Web 2.0 [11]. The idea is to utilize a semantic data in order to realize a more efficient approach to discover and make use of distributed services in the network. The semantic data are attached to the content of the web. Unlike syntactical information, these data are in a form that is understandable by the machine. In other words, it provides a way for the machines to communicate with one another so as to infer knowledge necessary for their decision making [18]. For example, a web search engine nowadays uses syntax of data such as keyword to find relevant websites.

One limitation of this technique is that the search engine could not distinguish elements that have the same symbolic name with different meanings. If one wants to search for the word “canine” with a sense of animal, the search engine will return all websites that publish contents of a canine in both an animal sense and physique sense because it could not realize a meaning of the word, “canine”. Thus, it depends on a human to manually filter out the irrelevant pages. With semantic web technologies, one could attach semantic data apart from the website content or service description (e.g. using an XML tag) so as to convey the meaning of the data (e.g. whether canine relates to either animal or an organ) in which the client application (e.g. search engine) can take an advantage of.

Specifically, meanings of symbols can be formed by noticing its relation with other symbols. For example, the canine has an animal sense (i.e. means “dog”) if and only if it can interact with other subjects (e.g. a canine can eat foods). Furthermore, the set of symbols that share the

same meaning can be categorized into a class in a similar way that scientific names are formed. This makes an essential ability that meaning of word can be carried to its subclass and constitutes a deeper meaning (e.g. a vegetarian restaurant is a restaurant that serves, in particular, vegetarian food). For example, the meaning of mammal animal is carried to the word canine. By using the property of mammal animal (i.e. a breast-feed animal) together with meaning of canine itself (a subject that has scientific name *Canis lupus familiaris*), one can infer the meaning of the symbol canine. Moreover, classification eases a task of semantic extraction. These are defined using ontologies. An ontology is different from XML schema in that the latter is defined a representation format but not convey any semantics [12].

There are various standards related to the semantic representation such as the semantic language to represent meaning, in particular, **Resource Description Framework (RDF)** and **Web Ontology Language (OWL)**. Moreover, there are software libraries used to retrieve information from the framework. Among others is a Java framework for building Semantic Web Application (**Jena**). These are the basis for the semantic web technologies and also serve as basic building blocks for this thesis (i.e. it is used for service discovery, service composition and also context retrieval). The core concepts will be elaborated throughout section 2.2.1 - 2.2.3.

2.2.1 Resource Description Framework (RDF)

The **Resource Description Framework (RDF)** [13] is a framework to describe ontology concepts which identified the resources (e.g. objects) and their relations on the web. The resources are identified by **Uniform Resource Identifier (URI)** and are classified into classes. The resources that belong to the same class share common properties. The properties have properties value which can combine with the resources to generate a statement in the form of a triple composed of subject, predicate and object. These triples act as key when one needs to search for a resource in RDF. An RDF is described in XML format as an example shown in Figure 24.

```
<?xml version="1.0"?>
<rdf:RDF
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:cd="http://www.recshop.fake/cd#">
  <rdf:Description
    rdf:about="http://www.recshop.fake/cd/Empire Burlesque">
    <cd:artist>Bob Dylan</cd:artist>
    <cd:country>USA</cd:country>
    <cd:company>Columbia</cd:company>
    <cd:price>10.90</cd:price>
    <cd:year>1985</cd:year>
  </rdf:Description>
  <rdf:Description
    rdf:about="http://www.recshop.fake/cd/Hide your heart">
    <cd:artist>Bonnie Tyler</cd:artist>
    <cd:country>UK</cd:country>
    <cd:company>CBS Records</cd:company>
    <cd:price>9.90</cd:price>
    <cd:year>1988</cd:year>
  </rdf:Description>
  .
  .
  .
</rdf:RDF>
```

Figure 24: An example of RDF document

Firstly, the `<rdf:RDF>` tag is a root of the document. It distinguishes the RDF document from other types of XML document. The `<rdf:Description>` tag represents a list of properties describing a resource defined in the attribute `<rdf:about>`. The properties can be given by primitive values (e.g. String), attributes or another resource. It is worth noting that RDF class is similar to an Object-Oriented programming class, in which one can define the class instance and its subclass that derives properties from its parent.

RDF schema is used in a way similar to XML schema. It is used to describe application specific classes and properties. Furthermore, a collection attribute can be used to define a limited set of properties that the resource can be bound to. An example of collection attribute is shown in Figure 25. In Figure 25, `rdf:about` is used to identify a resource. Apart from `rdf:about`, `rdf:id` tag could also be used to identify the resource. The difference between `rdf:about` and `rdf:id` is explained in [14].

```
<?xml version="1.0"?>
<rdf:RDF
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:cd="http://recshop.fake/cd#">
  <rdf:Description
    rdf:about="http://recshop.fake/cd/Beatles">
    <cd:artist rdf:parseType="Collection">
    <rdf:Description rdf:about="http://recshop.fake/cd/Beatles/George"/>
    <rdf:Description rdf:about="http://recshop.fake/cd/Beatles/John"/>
    <rdf:Description rdf:about="http://recshop.fake/cd/Beatles/Paul"/>
    <rdf:Description rdf:about="http://recshop.fake/cd/Beatles/Ringo"/>
    </cd:artist>
  </rdf:Description>
</rdf:RDF>
```

Figure 25: An example of Collection Attribute

RDF provides a simple way to describe ontologies that can be visualized using an RDF graph. The graph consists of a sphere denoted a subject, an arc denoted a property and a square denotes an object. Hence, the RDF graph completes a figure of RDF statement consisting of the triples subject, predicate and objects.

A resource is a container if it contains other resources or a limited set of primitive values. The container type has a subtype of BAG, ALT or SEQ. A BAG container represents a set of unordered items. An ALT container also represents a set of unordered items that is an alternative of one another. A SEQ container represents a set of ordered items. An example of RDF container represented by an RDF graph is shown in Figure 26. The resource has `rdf:type` property that is one of the `rdf:Bag`, `rdf:Alt`, `rdf:Seq` class or one of their subclasses. It also consists of properties `rdf:_1` up to `rdf:_n` to embody each member in the container where `n` is the number of members.

Despite the fact that RDF provides an effective way to express ontologies, it lacks functionality to depict complex relationships among classes. For example, it lacks functionality to describe an inverse relation. An inverse relation is a two-ways relationship that binds two properties together (e.g. a property `hasChild` and `hasParent` is an inverse relation. If a subject `P1` has child `P2`. Inversely, a subject `P2` has parent `P1`). For example, one can think of two properties, “is consist of” and “is part of”. A pizza “is consists of” pizza topping [16] and, in turn, the pizza topping is part of the pizza. The **Web Ontology Language (OWL)** is introduced in order to compensate this. We will elaborate on the OWL language in section 2.2.2. Indeed, all ontologies defined in this thesis are based on OWL (because it is easier if we want to extend ontology

with complex relationships (that RDF does not support) in future). However, as contexts in KMF are exchanged using RDF graph, it is worth understanding its basic notions.

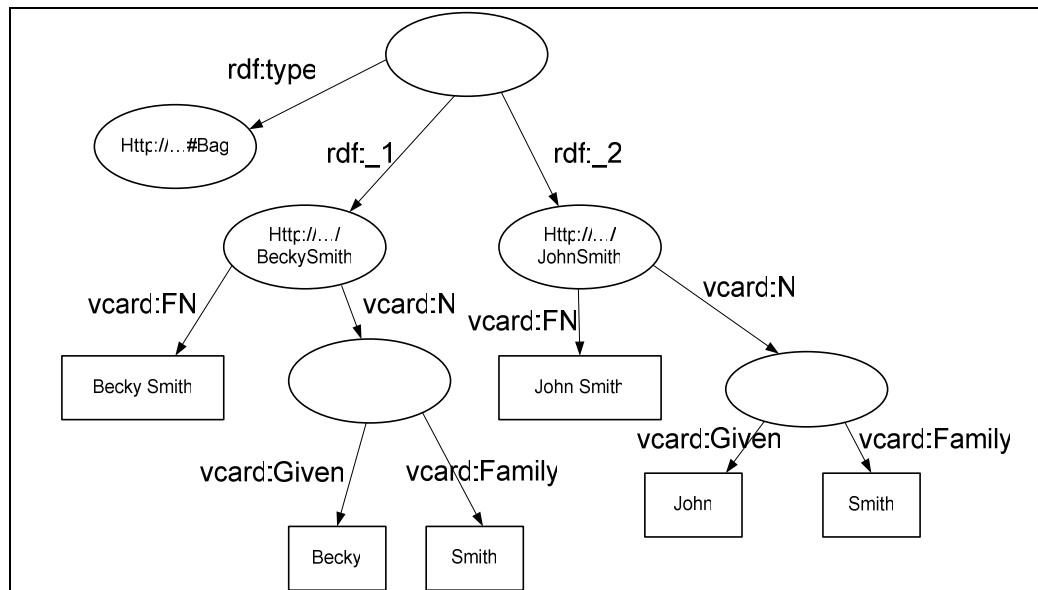


Figure 26: An example of RDF container

2.2.2 The Web Ontology Language (OWL)

The **Web Ontology Language (OWL)** [12] is a machine interpretable language that extends RDF capability. OWL has all RDF capabilities with some more functionality. For example, OWL supports functionality to define disjoint classes: No individual allows being members of both classes that are disjointed.

As RDF, OWL is expressed in XML format. Thus, it is independent of the programming language and operating system (platform-independent). OWL has three sublanguages, namely: OWL-Lite, OWL-DL and OWL-Full, which serves as web standards. Among the three languages, OWL-Lite is the simplest language and easy to use. However, it lacks some expressiveness that resides in OWL-DL and OWL-Full. OWL-Full has the most expressive power but, due to its complexity, the reasoning process (e.g. a semantic analysis) of the language is a time-consuming process and it is not practical in a system that has constrained resource. OWL-DL is the most optimal one and therefore it is normally deployed (in fact, we have used OWL-DL in our thesis).

Basically, OWL has three main components: Individuals, Properties and Classes [12]. As in a case of RDF, class is a basis for defining a relationship between elements. OWL has six types of classes: Named Class, Intersection Class, Union Class, Complement Class, Restriction and Enumerated Class. Apart from the first one which is a primitive class, they are complex classes. The complex classes can be constructed using a set of operators such as `intersectionOf`, `unionOf` and `complementOf`. These operators are similar to mathematical set operations. For example, a white wine class results from an intersection between a wine class and a class of things that is white in color [15]. A fruit class results from a union of a class of sweet fruits and a class of non-sweet fruits. A class of consumable things results from a complement of a class

Related technologies

of non-consumable things. An enumerated class defines a class that is bound to a limit set of values. For example, a wine color class has a value of white, rose or red [15].

Restrictions and Properties can be used to extend meanings of the classes. The restriction is used to restrict the range of a property. There are three main restrictions: Quantifier Restrictions, Cardinality Restrictions and Has-Value Restrictions. There are two types of quantifier restrictions that are *allValuesFrom* and *someValuesFrom*. *allValueFrom* defines a range for all property values. For example, for all wines, if they have makers, all the makers are wineries [15]. *someValueFrom* defines a range for some property values. For example, for all wines, they must have at least one maker which is winery [15]. The Cardinality Restriction defines a number of elements that a relation can be established. A *maxCardinality* provides an upper bound whereas a *minCardinality* provides a lower bound. A Has-Value restriction is used to classify a class based on property values that can be assigned to a class. For example, a Burgundy wine must have hasSugar property value equals to dry [15].

There are two main types of Properties: **Object properties** and **DataType properties** [12]. Object properties define a relationship between classes whereas DataType properties define a type of value that can be assigned to the classes. Object properties are assigned with domain and range. The domain identifies a class that the object property belongs to while the range identifies another class where the property is formed. For example, Object property, hasTopping, has a domain Pizza class and range PizzaTopping class [16].

Normally, classes are overlapping. This means that if two or more classes share the same property, they will be inferred to be identical. For example, ice-cream class could be inferred to be a pizza class as it poses a hasTopping property [16]. To avoid this effect, one could declare a class to be disjointed with other classes.

Classes are defined with set of conditions or concepts. The conditions convey information about the restrictions such as quantifier (e.g. for some \exists , for all \forall) that associate the property with other classes. For example, ice-cream class has a topping (i.e. hasTopping) for some fruit-topping (i.e. FruitTopping class) [16]. The conditions can be defined to be either “necessary” or “necessary and sufficient”. Classes possessing only necessary conditions are referred as primitive classes. Classes possessing one or more “necessary and sufficient” conditions are referred as defined class. A primitive classes could not subsume (i.e. a super class of) other classes. An example of OWL concept definition is shown in Figure 27.

<p>OWL: Class(CheesyPizza complete Pizza restriction (hasTopping someValuesFrom Cheese))</p> <p>Paraphrase: A cheesy pizza is <i>any</i> pizza that has, <i>amongst other things</i>, <i>some</i> cheese topping.</p>
--

Figure 27: An example of OWL concept definition

A class in OWL is not bound to a single scope. This implies that a class defined in one ontology file in one domain (e.g. for a specific application) is also valid on another domain. Moreover, the class can be referred in another ontology files using different name. In such a case, one declares an equivalence of two classes and properties using *equivalenceClass* and *equivalenceProperty* attributes (in fact, we have used this concept in defining synonym ontology. Two ontology classes (e.g. “find” and “search” are declared to be equivalence if they are syno-

Related technologies

nym of each other)). In the same way, an equivalence of two instances can be identified using *sameAs* attribute. In contrast, two instances can be distinguished using *differentFrom* and *All-Difference* attributes.

As mentioned in section 2.2.1 and also supports by [17], OWL extends capabilities of RDF in that it provides more expressive power in describing object properties. A property is transitive if and only if $P(x,y)$ and $P(y,z)$ implies $P(x,z)$. For example, if area x resides in area y and area y resides in area z then area x resides in area z . A property is symmetric if and only if $P(x,y) \Leftrightarrow P(y,x)$. For example, if area x is located near area y then area y is also located near area x . A property is a functional property if and only if $P(x,y)$ and $P(x,z)$ implies $y = z$. For example, a food has one and only one expiry date. A property $P1$ and $P2$ is an inverse property of one another if and only if $P1(x,y) \Leftrightarrow P2(y,x)$. For example, a pizza has a topping of pizza topping while a pizza topping has a base of pizza [16]. A property is an inverse functional property if and only if $P(y,x)$ and $P(z,x)$ implies $y = z$. For example, a wine has one and only one wine maker while wine maker produces one and only one wine [15]. It is worth noting that OWL does not support a capability to reason about *DataType* properties (e.g. literal values). For example, one could not describe that 7 is a prime number but could express that prime number has 7 as its member.

In fact, OWL is an integral part of this thesis as it is used to define goal ontology, input/output ontology and synonym ontology (that are used during a process of service discovery and service composition). Goal ontology keeps relationship between service goals. For example, in Figure 28, it indicates that the two goals, *GetGeoDistance* and *GetDistance* are similar. Moreover, *GetDistance* is a general goal while *GetGeoDistance* is a specific goal. This is because *GetGeoDistance* is a subclass of *GetDistance* in the ontology. Goal ontology is used by IA during a process of service discovery. Specifically, IA searches for services that have service goal (as defined in goal ontology) corresponding to the keywords in user request. We defined the goal ontology based on a class relationship (e.g. *subClassOf*). This is sufficient for our demonstration purpose. It is worth noting, however, that more complex property (as presented in this section) can be used to define a complex relationship between service goals.

```
<owl:Class rdf:ID="GetGeoDistance">
  <rdfs:subClassOf>
    <owl:Class rdf:ID="GetDistance"/>
  </rdfs:subClassOf>
</owl:Class>
```

Figure 28: An example of ontology class and its relation with another ontology class in goal ontology

```
<owl:Class rdf:ID="AirportInfo">
  <rdfs:subClassOf>
    <owl:Class rdf:ID="Text"/>
  </rdfs:subClassOf>
</owl:Class>
```

Figure 29: An example of ontology class and its relation with another ontology class in input/output ontology

```
<owl:Class rdf:ID="Get">
  <rdfs:subClassOf rdf:resource="http://www.w3.org/2002/07/owl#Thing"/>
  <owl:equivalentClass>
    <owl:Class rdf:about="#Find"/>
  </owl:equivalentClass>
  <owl:equivalentClass rdf:resource="#Search"/>
</owl:Class>
```

Figure 30: An example of ontology class and its relation with another ontology class in synonym ontology

Input/output ontology keeps relationship between service inputs/outputs. For example, in Figure 29, it indicates that the two service inputs/outputs, *AirportInfo* and *Text* are similar. In particular, *AirportInfo* can be used as *Text* but not vice versa. This is because *AirportInfo* is a subclass of *Text* in the ontology. Input/output ontology is used by IA during a process of service composition. Specifically, two services can be connected if output (or input) of one service can be used as input (or output) of another service. Similar to goal ontology, we defined the input/output ontology based on a class relationship (e.g. `subClassOf`). This is sufficient for our demonstration purpose. It is worth noting, however, that more complex property (as presented in this section) can be used to define a complex relationship between service inputs/outputs.

The synonym ontology keeps a relationship between words. For example, it tells that “find” and “search” have the same meaning. Synonym ontology is important because it provides flexibility to the end user. In other words, end user can choose the words he prefers when he requests for a service. For example, he can request either “**find** restaurant information in Paris” or “**search** restaurant information in Paris”. These two requests are identical so long as the synonym (find and search) are defined in synonym ontology. In Figure 30, it indicates that “find”, “get” and “search” are synonym of each others. This is because they are defined as equivalence class (using `equivalenceClass` property) in the ontology. We defined the synonym ontology based on a class relationship (e.g. `subClassOf`, `equivalenceClass`). This is sufficient for our demonstration purpose. It is worth noting, however, that more complex property (as presented in this section) can be used to define a complex relationship between synonym words.

Lastly, there are some rules that we followed while we defining the ontologies [19]. Specifically, we ensured that:

- The class hierarchy is correct.
- The class cycles are avoided.
- All sibling classes have the same level of generality.
- Additional intermediate classes will be added if there are more than a dozen subclasses for a specific class.
- A subclass have additional properties and restrictions that its superclass does not have.
- An instance of a class is the most specific concepts represented in any ontology.
- The concepts will be represented as classes if they form a natural hierarchy.

2.2.3 A Semantic Web Framework for Java (Jena)

Related technologies

Jena [20] is a Java API that is used to create and manipulate model (i.e. RDF graph) as the one shown in Figure 31. The RDF graph comprises of resources, properties and properties values (i.e. literals) that are represented by Jena classes: Resource, Property and Literal, respectively. The properties values can be other resources. Each arc in RDF graph is called RDF statement. As aforementioned, the RDF statement comprises of subject, predicate and object. The predicate is a label on the arc. The subject is a source of the arc and the object is the element that the edge of the arc points to.

Jena provides function calls to query model. Among them are *Model.getResource(String uri)* which returns a resource object corresponding to a resource URI, *Resource.getProperty(Property p)* which returns a statement associated with the resource and property, *Model.listStatements()* which returns a list of all available statements, *Model.listSubjects()* which returns a list of all available subjects, *Model.listSubjectsWithProperty(Property p,RDFNode o)* which returns a list of subjects that have property p with value o and *Model.listStatements(Selector s)* which returns a list of all statements that satisfy the selector condition (i.e. the one that satisfies the triple, subject, predicate and object). Moreover, more complex selector can be derived from simple selector to provide more functionality in filtering the set of statements. Jena provides three mathematical set operations, namely union, intersection and difference to construct one model from set of other models.

Apart from RDF, Jena supports OWL. There are two perspectives when referring to OWL. The first view treats OWL as a separate language and is not contingent to RDF. The second view treats OWL as RDF inheritance and considering the triple, predicates, subjects and objects, as a core part of OWL. Jena preserves the second view rather than the first.

The Jena Ontology API is not language specific. For example, the *OntClass* can be used to support any ontology languages such as OWL, RDFS [20] or DAML [20] depending on a profile parameterized to *OntModel* class. *OntModel* class represents an ontology model. The profile represents objects that are supported by the ontology language such as classes, properties and individuals. All state information is kept in the form of the triple (i.e. predicates, subjects and objects). The Java access methods of *OntClass* correspond to the predicate in an ontology model. An operation on *OntClass* does not modify the object itself but rather it modifies elements in the statement that are part of the ontology model.

The Jena API provides a support for reasoning about the ontology concepts through **inference API** (apart from the inference API, one could query ontology model through, a query language such as SPARQL as presented in [21] and [22]). A reasoning process is a process to derive a new concept from existing concepts. For example, if Fred is a fish then Fred is an animal. In fact, by using a reasoning mechanism, a new model (e.g. RDF model) can be created from an existing model. This new model possesses all concepts in term of triples that the old model has plus some new concepts. This implies that the new model can be used in place of the old model.

The Jena API provides a **Graph** interface to represent sets of RDF triples. The Graph interface comprises of base graph that holds asserted statements such as axioms. The *OntModel* can be used with or without a reasoner mechanism as it accesses the set of concepts via Graph interface. If the reasoner mechanism had been deployed, the new concepts would also be represented via Graph interface. Thus, the Graph interface encapsulates the computational mecha-

nism and hides inference complexities making the *OntModel* more abstract. This idea is depicted in Figure 32.

The RDF resource does not have a one-to-one mapping with an abstract class. For example, at one point in time, the resource could map to an *OntClass*. At another point, the resource could map to *Restriction class* (which is a subclass of *OntClass*) giving that the RDF model state has been changed. To solve this problem, Jena API utilizes a concept of facets. This means that a given RDF resource can have multiple facets associated with it at different time variances. The facets can be assigned to the RDF resource via a method call, *as()*. For example, *r.as(OntClass.class)* where *r* is an instance of *Resource* class.

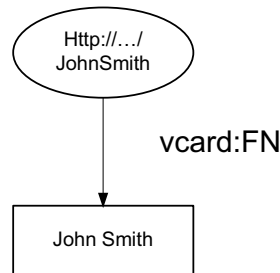


Figure 31: An example of RDF graph

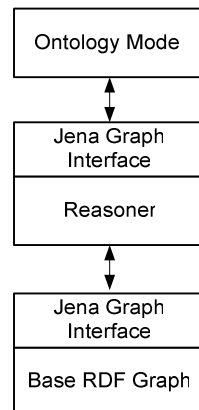


Figure 32: The Statement seen by the OntModel

The Jena API provides a support for manipulating an ontology via an *OntModel* interface. The *OntModel* can be constructed by invoking a factory method by passing a model specification as an argument. The model specification provides, for instance, a description about the language (e.g. RDFS, OWL Full, OWL DL, OWL Lite), the memory storage and the reasoner to be used. More sophisticated specification can be parameterized to *OntModel* as an *OntModelSpec* object. If there is no specification assigned to the factory method (i.e. *ModelFactory.createOntologyModel*), default specification will be used. The default specification for a language, memory usage and inference are OWL-Full language, in-memory storage and RDFS inference respectively.

An ontology value (e.g. any elements in the RDF triple) is represented by *OntResource* which extends Jena's RDF resource interface. Hence, any methods that accept RDF node also accept the *OntResource*. There are two types of ontology resource, namely, the asserted type and the

inferred type. The asserted type is the one that is explicitly attached with the resource. The inferred type is the one that is implicitly derived from a relationship. For example, if resource *x* has *rdf:type* of class *B* which is a subclass of *A*, *B* will be an asserted type while *A* will be an inferred type. Nevertheless, the inferred type can only be deduced via reasoning mechanism. In fact, a direct relationship can be built from asserted resource types. For instance, if *x* has *rdf:type* of class *B* which is a subclass of *D* and it also has *rdf:type* of class *D*, the direct inferred relationship can be established in a way that *x* has only *rdf:type* of class *D* because the former relationship (i.e. relationship with class *B*) can be inferred from the later (i.e. relationship with class *D*). The direct inferred relationship provides a more compact way to describe the type.

An ontology concept is represented through *OntClass*. There are two ways to get *OntClass*. The first method is to convert from RDF resource via the *as()* method as describe above. The second method is to get the ontology class from *OntModel* using its URI as reference. This method is achieved through a method call *getOntClass()* or *createClass()* (in fact, *getOntClass()* is used in this thesis because the name of ontology class (i.e. service goal) is known in advance). The difference is that the *createClass()* will create a new ontology class if the class could not be found in the ontology while the *getOntClass()* will return a null object. Jena API represents property via *OntProperty* class. The *OntProperty* is inherited further to support *ObjectProperty* and *DatatypeProperty* (there are other types of property, *AnnotationProperty*, with have no semantic meaning and only used for annotated purpose). The *ObjectProperty* is inherited to form a complex property relation such as *FunctionalProperty*, *TransitiveProperty*, *SymmetricProperty* and *InverseFunctionalProperty*.

The Jena API offers **list facilities** to entail all the classes defined in ontology. For example, a *listClasses()* method that returns an iterator that is used to list all the existing classes. The Jena API supports an *Individual* class to represent a member of class. The method *createIndividual()* taking ontology class as parameter creates an individual that belongs to the class. The *Individual* class provides methods to test and manipulate the class that individual fits in. The Jena API offers an *Ontology* class to manipulate metadata in an ontology file. The metadata is a set of attributes attached to the corresponding ontology file such as an author name, ontology version and the import statement. One can refer to a location of the metadata via a base URI of the ontology (a shorthand for referring to the base URI in ontology file is `<owl:Ontology rdf:about=" ">`).

As aforementioned, the reasoning mechanism can be attached to the ontology model. One can achieve this by invoking some ontology model specification method such as *setReasoner()* and *setReasonerFactory()*. The difference between the two is that the former supports only one ontology model (i.e. attaches a reasoner to a particular model) while the later supports multiple ontology models by mapping a set of reasoner to a set of models. The reason for attaching reasoning mechanism is to enrich the knowledge on the corresponding ontology. For example, if one knows that *A* is a father of *B*, one could infer that *A* is a parent of *B* as the reasoning mechanism can deduce that *fatherOf* is a sub-property of *parentOf*. In essence, the reasoning mechanism adds more set of triples to the ontology model. However, sometimes the reasoning mechanism adds a set of implications to the model without explicitly insert a new set of triples. One could access to this set of implications through *InfModel* interface which is accessible to the *OntModel* class. Considering that the ontologies developed in this thesis do not contain complex relationship (that need inference), we have not attached the reasoning mechanism to ontology model (rather, we reasoned about ontology concepts based on class relationships).

2.3 Service Creation Environment (SCE)

SPICE project has developed two environments: a **Service Creation Environment (SCE)** and a **Service Execution Environment (SEE)** [3]. An objective of SCE is to create and compose heterogeneous web services and Telecommunication services. An objective of SEE is to execute compositions of the services created by SCE. Hence, one of the goals of the SPICE SCE is to facilitate service compositions by supporting a formal user request. The formal user request is consisted of different kinds of semantic annotations such as functional properties (e.g. service input, output and goal) and non-functional properties (e.g. service qualities).

SCE and SEE fulfills a shortcoming in typical service description languages such as **Web Service Description Language (WSDL)**. SCE and SEE provide a mechanism to specify important non-functional service properties or quality attributes (such as service cost, performance metrics (e.g. response time), security attributes, reliability, and availability) of service composition [3].

In this section, we will provide a brief discussion on main components of SCE. Specifically, we will discuss about ACE that is a main component of SCE. ACE is a component that IA developed in this thesis interacts with. Then, we will discuss about SPATEL that is a service description language that ACE generates. The service composition expressed in SPATEL will be invoked (executed) by SPATEL execution engine of SEE. It is worth noting, however, that the thesis focuses on SCE rather than SEE (section 1.4).

2.3.1 Automatic Composition Engine (ACE)

The **Automatic Composition Engine (ACE)** [3] dynamically constructs service compositions that match a request issued by service developers. The request has to be constructed in a formal way to be useful by the ACE. The formal request consists of **functional property** and **non-functional property**. Functional property describes a characteristic of the desired service composition. Specifically, functional property describes:

- what kinds of service input (e.g. city name or country name) that the service composition takes
- what kinds of service output (e.g. weather information) that the service composition produces
- what objectives of the service composition are
- what objectives of the start node are
- what objectives of the end node are
- what alternative objectives are
- what kinds of consequence that the service composition generates apart from the service output
- what a prerequisite to invoke the service composition is. For example, what is an initial state of the service composition if that service composition comprises of stateful service components

Related technologies

Among others, the functional properties, service goals, start nodes and end nodes, are mandatory. Every formal request must have these functional properties associated with it. Non-functional property describes criteria of choosing service compositions (note that there could be more than one service compositions that are matched with functional property). For example, it describes what the minimum cost of service composition should be. Non-functional property is optional. However, it provides a hint that the ACE can use when it constructs the service compositions.

The formal request is constructed as follows:

- The service developers select ontology domains in which the service components are located.
- The service developers associate the functional properties with concepts defined in the domain ontology.
- The service developers associate the non-functional properties with concepts defined in the domain ontology.

Syntax of the formal request is shown in Figure 33.

```
<Ontologies>
<" domain of ontologies to be used for this service composition">
</Ontologies>
<Input>
  <"ontology concept of service input_1" name="name of service input_1">
  <"ontology concept of service input_2" name="name of service input_2">
  <"ontology concept of service input_3" name="name of service input_3">
  <"ontology concept of service input_n" name="name of service input_n">
</Input>
<Output>
  <"ontology concept of service output_1" name="name of service output_1">
  <"ontology concept of service output_2" name="name of service output_2">
  <"ontology concept of service output_3" name="name of service output_3">
  <"ontology concept of service output_n" name="name of service output_n">
</Output>
<Preconditions>
  <"ontology concept of precondition_1" name="name of precondition_1">
  <"ontology concept of precondition_2" name="name of precondition_2">
  <"ontology concept of precondition_3" name="name of precondition_3">
  <"ontology concept of precondition_n" name="name of precondition_n">
</Preconditions>
<Effects>
  <"ontology concept of effects_1" name="name of effects_1">
  <"ontology concept of effects_2" name="name of effects_2">
  <"ontology concept of effects_3" name="name of effects_3">
  <"ontology concept of effects_n" name="name of effects_n">
</Effects>
<Goal>
  <"ontology concept of service goal_1">
  <"ontology concept of service goal_2">
  <"ontology concept of service goal_3">
  <"ontology concept of service goal_n">
</Goal>
```

```
<Startnodes>
  <"ontology concept of service goal_1">
  <"ontology concept of service goal_2">
  <"ontology concept of service goal_3">
  <"ontology concept of service goal_n">
</Startnodes>
<Endnodes>
  <"ontology concept of service goal_1">
  <"ontology concept of service goal_2">
  <"ontology concept of service goal_3">
  <"ontology concept of service goal_n">
</Endnodes>
<Optionals>
  <"ontology concept of service goal_1">
  <"ontology concept of service goal_2">
  <"ontology concept of service goal_3">
  <"ontology concept of service goal_n">
</Optionals>
<Non-functional>
  <"ontology concept of non-functional property" value="value of non-functional property">
</Non-functional>
```

Figure 33: Syntax of formal request sent to ACE

Inside Ontologies tag, we specify a list of ontology domains (e.g. URI that identifies that ontology domains) to be used for the service composition. Inside functional property tag, such as input tag, we specify ontology classes (e.g. in a form of URI) indicating the resource (e.g. service input) and the name of that resource. The ontology classes have to be valid resources defined in one of the ontology domains. The name can be any valid name but it must not be duplicated with names of other resources in the tag. The names will be used for linking service components. We do not have to specify name for the goal tag as a purpose of goal tag is solely to indicate service components to be used (and not link them). Inside start node's and end node's tag, we specify ontology classes indicating the goals of start node and end node, respectively. We also specify ontology classes indicating the optional goals between optional tags.

Inside non-functional property tag, we specify ontology classes (e.g. in a form of URI) indicating the resource (e.g. a criterion) and its value. An example of formal request for *"receives a piece of text, translates it to English, and sends the resulting text by SMS to a given destination number"* is shown in Figure 34.

```
<Ontologies>
<"GoalOnt" "TelecomOnt" "NFPont" "LanguageOnt">
</Ontologies>
<Input>
  <"LanguageOnt#Language" name="srcLang">
  <"LanguageOnt#English" name="trgtLang">
  <"LanguageOnt#Text" name="txtToTrans">
  <"TelecomOnt#PhoneNum" name="destNumber">
</Input>
<Output>
  <"TelecomOnt#AckSMS"
    name="AcknowledgementSMS">
</Output>
```



```
<Preconditions/><Effects/>
<Goal>
  <"GoalOnt#translate">
  <"GoalOnt#sendSMS">
</Goal>
<Startnodes>
  <"GoalOnt#translate">
</Startnodes>
<Endnodes>
  <"GoalOnt#sendSMS">
</Endnodes>
<Optionals/>
<Non-functional>
  <"NFPont#Cost" value=6>
</Non-functional>
```

Figure 34: An example of formal composition request

ACE is composed of three main components, namely, **Composition Factory** [3], **Non-Functional Property Aggregation and Matcher** [3]. These components form **Composition Process Module** [3]. Composition Factory provides service discovery functionality and performs the service composition if the service request was not found in service repository. Non-Functional Property Aggregation accumulates the non-functional properties (e.g. response time, cost of the service) of service components. Matcher ensures that the composed service matches with the desired service component and ranks the service composition candidates. The ranking list of service composition candidates are delivered to the user (e.g. service developer) for further validation. Moreover, there are additional supportive modules: **Semantic Reasoning Module** [3], **Service Discovery and Selection Module** [3], **Causal Link Matrix (CLM⁺) Construction module** [3] and **Graph based Composition Module** [3]. Semantic Reasoning Module provides semantic inference capabilities that based on analyzing concepts defined in ontologies. The CLM⁺ construction module utilizes Semantic Reasoning Module to infer a relationship between service components. The Graph based Composition module constructs a service composition by using information gathered from CLM⁺ construction module. Finally, the result of Graph based Composition module is translated to **SPATEL**. We will discuss about SPATEL in the subsequent section.

One challenge of this thesis is to extract the semantic information from informal requests that are in natural language. Specifically, we analyze user request in order to extract service goals from the sentence. From service goals, we derive other semantic information such as service inputs and outputs. Based on relationships between service inputs and outputs, we derive start nodes and end nodes. In our scenarios, we did not concern about the ontology domains we used. This is because we assumed, for simplicity, that the user request always associated with two ontology domains. That is, the service goal always associates with a domain of goal ontology (this ontology is indeed shared between ACE and IA, Figure 3). Furthermore, the service input/output always associates with a domain of input/output ontology (this ontology is indeed shared between ACE and IA, Figure 3).

It is worth noting that IA, that we developed, also provides other exclusive information to ACE. Specifically, IA provides information about how services should be composed. For example, if we have three services, WeatherReport, GoogleTranslate and SendSMS, IA will inform ACE that the order of composition is WeatherReport->GoogleTranslate->SendSMS. IA realizes the order of composition by noticing a relationship between service inputs and outputs. The order of composition will also be verified and corrected by

using NLP technique in order to get the most accurate result. With this information, ACE can easily generate service composition in SPATEL. However, as shown in

Figure 33, the current version of ACE does not provide an interface for IA to inform this information through the formal request. The prototype version of ACE (that we experimented with) has been modified to take this information.

Moreover, the prototype version of ACE only receives a formal request through a file (that is, we have to put a formal request into a file, that ACE is configured to read). Due to time limitation, we take a simple approach to connect ACE with IA. That is, we integrate the two components in a source code level (i.e. passed formal request via Java object). In future work, the interface between ACE and IA should be done by using efficient means such as web services.

2.3.2 SPICE Advance Language for Telecommunication Services (SPATEL)

SPATEL [42] is a high-level language that is used for designing service compositions. The service composition described in SPATEL is presented through state machines [43]. SPATEL can be used by a service developer to define manually the logic of a service composition. It can also be used by ACE to produce the logic of the composition based on a formal request received from service developer. SPATEL concepts are represented by OWL ontology. The concepts are related to service design notions such as *ServiceInterface*, *ServiceOperation*, *ServiceEvent*, *ServiceContract*, *ServiceUsage*, *Dialogs* and so on [42]. As SPATEL represents concepts in a form of ontology, the applications (such as SPATEL execution engine) that exploit SPATEL descriptions can use ontology techniques to reason on SPATEL definitions. In the scenarios where automatic discovery and selection of services are needed, SPATEL also provides a mechanism to embed various kinds of semantic and **Quality of Service (QoS)** annotations.

The syntax and semantic of SPATEL can be shown, by example, as in Figure 35 and Figure 36. `<ServiceLibrary>` contains XML representation of service composition. All services are packaged under `<nestedPackage>` (in the example, there is only one service that is composed from several service components). The `<service>` indicates the service composition being composed. `<ownedOperation>` indicates an operation of the service composition (that is, to “orchestrate” service components). `<behavior>` specifies details of the service operation. `<initSection>` represents proxies[43] and variables to be created in an initial state of the state machine [43]. Proxy is a representative of service operation. Variable is a slot to store result values when an operation invocation occurs. The state machine of this service composition is represented in `<region>`.

Each `<region>` specifies a graph of vertexes (nodes in the state machine, represented by `<sub-vertex>`) and transitions (represented by `<transition>`). There are two kinds of vertexes: states [43] and pseudo-states [43]. States are categorized into simple type (representing simple unique region), composite type (representing more than one region) and submachine type (representing a contained state machine). Pseudo-states are abstraction states encompassed different types of transient vertexes.

Transition is an element leads from one state (i.e. vertex) to another state. Transition consists of trigger, guard and effect [43]. States will be changed when trigger (representing event [43]) is activated and guard (representing condition) satisfies. Effect (not shown in the example) indicates a sequence of actions to be invoked if the states are changed.

Related technologies

There are two kinds of actions (or events). Accept action [43] is a specific kind of action that occurs after a wait state. The wait state represents a stable situation for receiving a given list of candidate input events. Event-sending action [43] is a specific kind of action that occurs within a transition linking two states.

<variable> indicates variables that are defined and used in the scope of state machine. Normally, values of variables are checked in guards and assigned in actions. Finally, <event> represents, for this case, the first event to be arisen before the state machine will transform from the first wait state.

In this section, we provided a brief discussion on SPATEL. We started from explaining SPATEL in general and exploring the SPATEL core syntax and semantic. It is worth noting that we have not represented every XML node or XML schema. Rather, we discussed only on XML nodes that are generated from this thesis. This is because this thesis does not focus on SEE. We refer to [43] for more detailed discussion on SPATEL.

```
-<xmi:XMI xmi:version="2.0">
  -<spatel:ServiceLibrary name="CompositeService" xmi:id="rgKwKvJlMlmi295WLqpnBCvJ">
    -<nestedPackage name="CompositeService" xmi:id="qA2SOqPqWw+wLCCcZ_1zPLhGg" xsi:type="spatel:ServicePackage">
      -<service name="CompositeService" xmi:id="OdeJmni8nB3TWUzXxulKHhORNo" xsi:type="spatel:ServiceInterface">
        -<ownedOperation name="orchestrate" xmi:id="Pph5151TPStp958tHElIf4e+wP" xsi:type="spatel:ServiceOperation">
          -<behavior xmi:id="apquqDBDca+azf478Y7XSZ+vKu" xsi:type="spatel:StateMachine">
            -<initSection>
              <action opaqueBody="var SEA : SearchVegetarianRestaurant = createProxy("SearchVegetarianRestaurant")" xsi:type="spatel:VariableDeclarationAction"/>
              <action opaqueBody="var IN1 : String" xsi:type="spatel:VariableDeclarationAction"/>
              <action opaqueBody="var SEN : SendSms = createProxy("SendSms")" xsi:type="spatel:VariableDeclarationAction"/>
              <action opaqueBody="var OUT1 : String" xsi:type="spatel:VariableDeclarationAction"/>
            </initSection>
            -<region>
              <subvertex kind="initial" name="Initial1" outgoing="Qfd5f40PXUuBdZy0yMjLMoqB6" xmi:id="xYPnqFCdCcUuae23Q_QMieh73" xsi:type="spatel:InitialNode"/>
              <transition name="tr1" source="xYPnqFCdCcUuae23Q_QMieh73" target="yFvJF9qB7XTWSprrLEaDdY+5n7" xmi:id="Qfd5f40PXUuBdZy0yMjLMoqB6"/>
              <subvertex incoming="Qfd5f40PXUuBdZy0yMjLMoqB6" name="Wait" outgoing="WOosoCAC8ZORspwnC9acdzc0+O" xmi:id="yFvJF9qB7XTWSprrLEaDdY+5n7"
                xsi:type="spatel:WaitState"/>
              -<transition name="tr2" source="yFvJF9qB7XTWSprrLEaDdY+5n7" target="DtTrEGHj0QMTtoEpD9Zge1c4y" xmi:id="WOosoCAC8ZORspwnC9acdzc0+O">
                -<trigger>
                  <acceptAction opaqueBody="GetValues(IN1)"/>
                </trigger>
                <trigger>
                </trigger>
                <transition>
                </transition>
                <subvertex incoming="WOosoCAC8ZORspwnC9acdzc0+O" kind="accept" name="GetValues(IN1)" outgoing="6_OKkNmvrFBD9dgh5kmhADDeVB"
                  xmi:id="DtTrEGHj0QMTtoEpD9Zge1c4y" xsi:type="spatel:AcceptNode"/>
                <transition name="tr3" source="DtTrEGHj0QMTtoEpD9Zge1c4y" target="WmBqEx_PQTOoltGCZCbZg2_2j" xmi:id="6_OKkNmvrFBD9dgh5kmhADDeVB"/>
              -<subvertex incoming="6_OKkNmvrFBD9dgh5kmhADDeVB" name="VAR1 = SEA.SearchVegetarianRestaurant(IN1)" outgoing="20PWSuprsHIEjImi29rGrEFbde"
                xmi:id="WmBqEx_PQTOoltGCZCbZg2_2j" xsi:type="spatel:SyncCallState">
                <callAction opaqueBody="VAR1 = SEA.SearchVegetarianRestaurant(IN1)" xsi:type="spatel:AssignmentAction"/>
              </subvertex>
              <transition name="tr4" source="WmBqEx_PQTOoltGCZCbZg2_2j" target="dloCAZ7dac0zL74VNrmDmABF" xmi:id="20PWSuprsHIEjImi29rGrEFbde"/>
              -<subvertex incoming="20PWSuprsHIEjImi29rGrEFbde" name="OUT1 = SEN.SendSms(VAR1,IN2)" outgoing="TVxsGf_xLPLqikBD5VRtSxYy_"/>
```

Figure 35: An example of SPATEL (1)

```

- <transition name="tr2" source="yFvJF9qB7XTWSsprLEaDdY+5n7" target="DtTrEGHj0QMTtoEpD9Zge1c4y" xmi:id="WOosoCAC8ZORspwnC9acdzc0+O">
- <trigger>
  <acceptAction opaqueBody="GetValues(IN1)"/>
</trigger>
</transition>
<subvertex incoming="WOosoCAC8ZORspwnC9acdzc0+O" kind="accept" name="GetValues(IN1)" outgoing="6_OKkNmvrFBD9dgh5kmhADDeVB"
xmi:id="DtTrEGHj0QMTtoEpD9Zge1c4y" xsi:type="spatel:AcceptNode"/>
<transition name="tr3" source="DtTrEGHj0QMTtoEpD9Zge1c4y" target="WmBqEx_PQTOoltGCZCbZg2_2j" xmi:id="6_OKkNmvrFBD9dgh5kmhADDeVB">
- <subvertex incoming="6_OKkNmvrFBD9dgh5kmhADDeVB" name="VAR1 = SEA.SearchVegetarianRestaurant(IN1)" outgoing="20PWSuprsHIEjImi29rGrEFbde"
xmi:id="WmBqEx_PQTOoltGCZCbZg2_2j" xsi:type="spatel:SyncCallState">
  <callAction opaqueBody="VAR1 = SEA.SearchVegetarianRestaurant(IN1)" xsi:type="spatel:AssignmentAction"/>
</subvertex>
<transition name="tr4" source="WmBqEx_PQTOoltGCZCbZg2_2j" target="dloCAZ7dac0zLi74VNRmDmABF" xmi:id="20PWSuprsHIEjImi29rGrEFbde"/>
- <subvertex incoming="20PWSuprsHIEjImi29rGrEFbde" name="OUT1 = SEN.SendSms(VAR1,IN2)" outgoing="TVxsGf_xLPLqkqBD5VRtSxYy_"
xmi:id="dloCAZ7dac0zLi74VNRmDmABF" xsi:type="spatel:SyncCallState">
  <callAction opaqueBody="OUT1 = SEN.SendSms(VAR1,IN2)" xsi:type="spatel:AssignmentAction"/>
</subvertex>
<transition name="tr5" source="dloCAZ7dac0zLi74VNRmDmABF" target="FcZ+Yyy+PLnKmpCuKtHFMme7g" xmi:id="TVxsGf_xLPLqkqBD5VRtSxYy_">
<subvertex incoming="TVxsGf_xLPLqkqBD5VRtSxYy_" name="Final" xmi:id="FcZ+Yyy+PLnKmpCuKtHFMme7g" xsi:type="spatel:FinalState"/>
</region>
<variable name="VAR1"/>
<variable name="SEA"/>
<variable name="SEN"/>
<behavior>
  <ownedOperation>
</service>
</service>
- <event name="GetValues" xmi:id="T6WSb_w-wKILMrQrn7EBcYbTx" xsi:type="spatel:ServiceEvent">
  <ownedAttribute instanceType="String" name="IN1" xmi:id="325VXZVz0yMKNJfb03_R589ard" xsi:type="spatel:ServiceAttribute"/>
</event>
</nestedPackage>
</spatel:ServiceLibrary>
</xmi:XMI>

```

Figure 36: An example of SPATEL (2)

2.4 Knowledge Management Framework (KMF)

Consider the following circumstance captured from a recent work [5] on defining the research challenges in mobile and context-aware service development:

Mr. Smith is a sales representative of a large company and is attending a business meeting where the economic analysis manager is representing the results of a recent survey and where the participants are discussing marketing strategies. All attendants can use a display to interact with the shared white board in the conference room. Mr. Smith has to leave the meeting early as he is scheduled to visit a client to address some long-standing maintenance problems. During the trip, he would like to stay in touch with his colleagues and discuss the proposals being presented.

The conference room is able to broadcast meetings for audience members who cannot contribute live to the meeting, but who are able to watch and listen to the presentation remotely. They can interact during the meeting using a text-based communication application to send and respond to questions or to read the text that the presenter enters. Upon detection that Mr. Smith is leaving the conference room, the conferencing client moves from his display in the conference room to his personal wireless handheld device. As the I/O capabilities of his handheld are rather limited, some parts of the conferencing client can temporarily disabled, replaced or moved to other devices. Parts of the client can also be moved to devices in the vicinity of Mr. Smith because the battery has no more power, a larger display is available...

The case illustrates, among other things, a requirement on an ability to detect the user settings such as a location of Mr. Smith and his device capability. Upon successful detection of the user contexts, the system can adjust its service accordingly. For example, when Mr. Smith location

is changed (e.g. when he is moving), the appropriate service to be delivered should be the one that supports handheld devices rather than fixed devices [5]. Another example is that Mr. Smith requests for news while he is driving. The appropriate service is to send an SMS containing the news to his mobile phone rather than to send news to the lease line fax machine [3]. Hence, given the settings, a system that supports context handling functionality provides an efficient way to deliver the most appropriate service to the user [23].

A **knowledge management framework (KMF)** [6] is developed in SPICE project to provide this functionality. It is worth noting that the knowledge management framework is also called a **context management framework (CMF)** [6]. This is due to a historical reason that this kind of framework has difference names in different projects. Thus, the word, KMF and CMF is interchangeable. In this chapter, the concept of KMF will be elaborated in further details. One of the challenges in the thesis is to integrate our work with the KMF platform so as to improve a response to the user request as the preceding example illustrated. In this section, we will elaborate on components in KMF and the functions they provide to support context handling.

2.4.1 Knowledge Management Framework (KMF): Overview

The **Knowledge Management Framework (KMF)** [6] is a middleware framework that supports a gathering and processing of context information (referred as knowledge) that is exchanged as instances of ontology concepts. An example of context ontology exchanged in KMF is illustrated in Figure 37. Figure 37 can be interpreted as follows:

The context is composed of three nodes. These are node A0, A1 and A2.

Node A1 indicates that the context is a location context (rdf:type = **UserLocation**). Moreover, it is a location of a user defined in node A0 (j.0:isLocationOf = A0). The location itself is defined in node A2 (j.0:isLocatedIn = A2). Also, the timestamp of this context is also defined in node A1 (j.0:timestamp).

Node A0 indicates that the context is of a user Remco.Poortinga@telin.nl (j.0:identifier = Remco.Poortinga@telin.nl). It also defines a type of this user through other context ontologies. For example, this user is a Person (rdf:type = Person) which can be considered as PhysicalEntity (rdf:type = PhysicalEntity). Note that the concept of Person, for example, is defined in <http://amigo.gforge.inria.fr/owl/Amigo.owl>.

Node A2 indicates that the location of the user is Canteen@telin.nl (j.0: identifier = Canteen@telin.nl). It also defines a type of this location through other context ontologies. For example, this location is a PhysicalEntity (rdf:type = PhysicalEntity) and it is a Space (rdf:type = Space).

Figure 37 illustrates a location context. It is worth noting that a security context is similar to location context with some exceptions. An example of an authentication/authorization context is shown in Figure 38. As shown in Figure 38, node A1 indicates that the context is an authentication/authorization context (rdf:type = **UserAuthentication**). Moreover, it gives a permission for an access to a service of a user defined in node A0 (j.0:isAuthenticationOf = A0). The service itself is defined in node A2 (j.0:isAuthenticatedTo = A2). Also, the timestamp of this

Related technologies

context is also defined in node A1 (j.0:timestamp). Node A2 indicates the service that the user is allowed to access.

An example of a restriction context is shown in Figure 39. As shown in Figure 39, node A1 indicates that the context is a restriction context. Notice that restriction context and authentication/authorization context belong to the same category (rdf:type = **UserAuthentication**). Moreover, it denies an access to a service of a user defined in node A0 (j.0:isAuthenticationOf = A0). The service itself is defined in node A2 (j.0:isRestrictedTo = A2). Also, the timestamp of this context is also defined in node A1 (j.0:timestamp). Node A2 indicates the service that the user is not allowed to access.

The main components of KMF are Knowledge Source, Knowledge Sink and Knowledge Broker. A Knowledge Source registers for its supplying knowledge (i.e. the type of knowledge that the Knowledge Source supports) with Knowledge Broker.

```
<rdf:RDF
  xmlns:Amigo="http://amigo.gforge.inria.fr/owl/Amigo.owl#"
  xmlns:AmigoDevices="http://amigo.gforge.inria.fr/owl/Devices.owl#"
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema#"
  xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
  xmlns:owl="http://www.w3.org/2002/07/owl#"
  xmlns:AmigoICCS="http://amigo.gforge.inria.fr/owl/AmigoICCS.owl#"
  xmlns:daml="http://www.daml.org/2001/03/daml+oil#"
  xmlns:dc="http://purl.org/dc/elements/1.1/"
  xmlns:j.0="http://amigo.gforge.inria.fr/owl/ContextTransport.owl#" >
  <rdf:Description rdf:nodeID="A0">
    <j.0:identifier rdf:datatype="http://www.w3.org/2001/XMLSchema#string">Remco.Poortinga@telin.nl</j.0:identifier>
    <rdf:type rdf:resource="http://amigo.gforge.inria.fr/owl/Amigo.owl#Person"/>
    <rdf:type rdf:resource="http://amigo.gforge.inria.fr/owl/Amigo.owl#PhysicalEntity"/>
    <rdf:type rdf:resource="http://amigo.gforge.inria.fr/owl/Amigo.owl#Entity"/>
    <rdf:type rdf:resource="http://amigo.gforge.inria.fr/owl/Amigo.owl#AmigoConcept"/>
    <rdf:type rdf:resource="http://amigo.gforge.inria.fr/owl/AmigoICCS.owl#User"/>
  </rdf:Description>
  <rdf:Description rdf:nodeID="A1">
    <j.0:timestamp rdf:datatype="http://www.w3.org/2001/XMLSchema#dateTime">2008-02-04T15:51:37.39Z</j.0:timestamp>
    <j.0:isLocatedIn rdf:nodeID="A2"/>
    <j.0:isLocationOf rdf:nodeID="A0"/>
    <rdf:type rdf:resource="http://amigo.gforge.inria.fr/owl/ContextTransport.owl#UserLocation"/>
  </rdf:Description>
  <rdf:Description rdf:nodeID="A2">
    <j.0:identifier rdf:datatype="http://www.w3.org/2001/XMLSchema#string">Canteen@telin.nl</j.0:identifier>
    <rdf:type rdf:resource="http://amigo.gforge.inria.fr/owl/Amigo.owl#PhysicalEntity"/>
    <rdf:type rdf:resource="http://amigo.gforge.inria.fr/owl/Amigo.owl#Entity"/>
    <rdf:type rdf:resource="http://amigo.gforge.inria.fr/owl/Amigo.owl#AmigoConcept"/>
    <rdf:type rdf:resource="http://amigo.gforge.inria.fr/owl/AmigoICCS.owl#Space"/>
  </rdf:Description>
</rdf:RDF>
```

Figure 37: An example of location context

```

<rdf:RDF
  xmlns:Amigo="http://amigo.gforge.inria.fr/owl/Amigo.owl#"
  xmlns:AmigoDevices="http://amigo.gforge.inria.fr/owl/Devices.owl#"
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema#"
  xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
  xmlns:owl="http://www.w3.org/2002/07/owl#"
  xmlns:AmigoICCS="http://amigo.gforge.inria.fr/owl/AmigoICCS.owl#"
  xmlns:daml="http://www.daml.org/2001/03/daml+oil#"
  xmlns:dc="http://purl.org/dc/elements/1.1/"
  xmlns:j.0="http://amigo.gforge.inria.fr/owl/ContextTransport.owl#" >
  <rdf:Description rdf:nodeID="A0">
    <j.0:identifieur rdf:datatype="http://www.w3.org/2001/XMLSchema#string">Remco.Poortinga@telin.nl</j.0:identifieur>
    <rdf:type rdf:resource="http://amigo.gforge.inria.fr/owl/Amigo.owl#Person"/>
    <rdf:type rdf:resource="http://amigo.gforge.inria.fr/owl/Amigo.owl#PhysicalEntity"/>
    <rdf:type rdf:resource="http://amigo.gforge.inria.fr/owl/Amigo.owl#Entity"/>
    <rdf:type rdf:resource="http://amigo.gforge.inria.fr/owl/Amigo.owl#AmigoConcept"/>
    <rdf:type rdf:resource="http://amigo.gforge.inria.fr/owl/AmigoICCS.owl#User"/>
  </rdf:Description>
  <rdf:Description rdf:nodeID="A1">
    <j.0:timestamp rdf:datatype="http://www.w3.org/2001/XMLSchema#dateTime">2010-05-28T14:00:44.964Z</j.0:timestamp>
    <j.0:isAuthenticatedTo rdf:nodeID="A2"/>
    <j.0:isAuthenticationof rdf:nodeID="A0"/>
    <rdf:type rdf:resource="http://amigo.gforge.inria.fr/owl/ContextTransport.owl#UserAuthentication"/>
  </rdf:Description>
  <rdf:Description rdf:nodeID="A2">
    <j.0:identifieur rdf:datatype="http://www.w3.org/2001/XMLSchema#string">GlobalSMS</j.0:identifieur>
    <rdf:type rdf:resource="http://amigo.gforge.inria.fr/owl/Amigo.owl#Service"/>
    <rdf:type rdf:resource="http://amigo.gforge.inria.fr/owl/Amigo.owl#PhysicalEntity"/>
    <rdf:type rdf:resource="http://amigo.gforge.inria.fr/owl/Amigo.owl#Entity"/>
    <rdf:type rdf:resource="http://amigo.gforge.inria.fr/owl/Amigo.owl#AmigoConcept"/>
  </rdf:Description>
</rdf:RDF>

```

Figure 38: An example of authentication context

```

<rdf:RDF
  xmlns:Amigo="http://amigo.gforge.inria.fr/owl/Amigo.owl#"
  xmlns:AmigoDevices="http://amigo.gforge.inria.fr/owl/Devices.owl#"
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema#"
  xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
  xmlns:owl="http://www.w3.org/2002/07/owl#"
  xmlns:AmigoICCS="http://amigo.gforge.inria.fr/owl/AmigoICCS.owl#"
  xmlns:daml="http://www.daml.org/2001/03/daml+oil#"
  xmlns:dc="http://purl.org/dc/elements/1.1/"
  xmlns:j.0="http://amigo.gforge.inria.fr/owl/ContextTransport.owl#" >
  <rdf:Description rdf:nodeID="A0">
    <j.0:identifieur rdf:datatype="http://www.w3.org/2001/XMLSchema#string">Remco.Poortinga@telin.nl</j.0:identifieur>
    <rdf:type rdf:resource="http://amigo.gforge.inria.fr/owl/Amigo.owl#Person"/>
    <rdf:type rdf:resource="http://amigo.gforge.inria.fr/owl/Amigo.owl#PhysicalEntity"/>
    <rdf:type rdf:resource="http://amigo.gforge.inria.fr/owl/Amigo.owl#Entity"/>
    <rdf:type rdf:resource="http://amigo.gforge.inria.fr/owl/Amigo.owl#AmigoConcept"/>
    <rdf:type rdf:resource="http://amigo.gforge.inria.fr/owl/AmigoICCS.owl#User"/>
  </rdf:Description>
  <rdf:Description rdf:nodeID="A1">
    <j.0:timestamp rdf:datatype="http://www.w3.org/2001/XMLSchema#dateTime">2010-05-28T14:00:44.964Z</j.0:timestamp>
    <j.0:isRestrictedTo rdf:nodeID="A2"/>
    <j.0:isAuthenticationof rdf:nodeID="A0"/>
    <rdf:type rdf:resource="http://amigo.gforge.inria.fr/owl/ContextTransport.owl#UserAuthentication"/>
  </rdf:Description>
  <rdf:Description rdf:nodeID="A2">
    <j.0:identifieur rdf:datatype="http://www.w3.org/2001/XMLSchema#string">GlobalSMS</j.0:identifieur>
    <rdf:type rdf:resource="http://amigo.gforge.inria.fr/owl/Amigo.owl#Service"/>
    <rdf:type rdf:resource="http://amigo.gforge.inria.fr/owl/Amigo.owl#PhysicalEntity"/>
    <rdf:type rdf:resource="http://amigo.gforge.inria.fr/owl/Amigo.owl#Entity"/>
    <rdf:type rdf:resource="http://amigo.gforge.inria.fr/owl/Amigo.owl#AmigoConcept"/>
  </rdf:Description>
</rdf:RDF>

```

Figure 39: An example of restriction context

Each Knowledge Source owns sub-profiles which contain an identifier that provides information about users. This sub-profile can only be requested by a service that is the owner of the profile or the one that has permission.

Basically, the main functions of a Knowledge Source are:

- to get knowledge from other Knowledge Source
- to process and derive a new knowledge from other knowledge sources

The main functions of Knowledge Broker are:

- to discover a Knowledge Source
- to provide an access to the Knowledge Source
- to maintain a registry containing a mapping between Knowledge Source and Knowledge type

The main functions of Knowledge Sink are:

- to enquire a Knowledge Broker for a Knowledge Source that maintains particular types of knowledge
- to connect to the Knowledge Source
- to retrieve the knowledge

The information that is transmitted in the KMF is called knowledge (or context). In this thesis, we concentrate on two types of context. These are location context and security context. It is worth noting that KMF does not receive input from external sources. Rather, the knowledge source itself established context information to the framework [6]. In practice, examples of knowledge source are sensors (that can detect locations of the user) and Bluetooth detectors. For simplicity, in this thesis, we have used a simple simulator of knowledge source for our demonstration purpose.

In the subsequent sections, we will elaborate on Knowledge Source, Knowledge Sink and Knowledge Broker which are the three important components of the framework that contributes to a context handling part of this thesis. Specifically, we have developed a Knowledge Sink as part of IA to retrieve user contexts from KMF.

2.4.2 Knowledge Source

As aforementioned, Knowledge Source [6] describes the knowledge that it supports to Knowledge Broker through OWL/RDF document (OWL and RDF are discussed in section 2.2). This information will be used when Knowledge Sink asks Knowledge Broker to search for Knowledge Source and also when the knowledge is exchanged among the sources.

There are three specialized types of Knowledge Source, namely, Context Wrapper, Knowledge Reasoner and Knowledge Storage. Context Wrapper acts as a container for knowledge derived from outside KMF. Knowledge Reasoner has a functionality to enrich its existing knowledge with other knowledge obtained from other Knowledge Source. Knowledge Storage stores knowledge about usage patterns. The different types of Knowledge Source are shown in

Figure 40. In this thesis, a simple type of Knowledge Source (not a specialized type) is sufficient for our demonstration purpose.

Knowledge Source can derive a new knowledge from other Knowledge Sources knowledge by utilizing theorem prover or backward reasoning mechanism (this is corresponding to the idea presented in [24]). This derivation process is called **Context Aggregation**.

According to [6], Knowledge Source provides multiple types of interface (these interfaces are used by Knowledge Sink to retrieve context). The *IContextSource* interface provides a synchronous access (discussed in section 2.4.3) to the knowledge that the Knowledge Source provides. The *IContextSubscription* interface provides an asynchronous access (discussed in section 2.4.3) to the knowledge that the Knowledge Source provides. In this thesis, we simulated a knowledge source and made it dynamically feed distinct contexts into the system (where IA is operated on) in order to demonstrate an effectiveness of our context-awareness application.

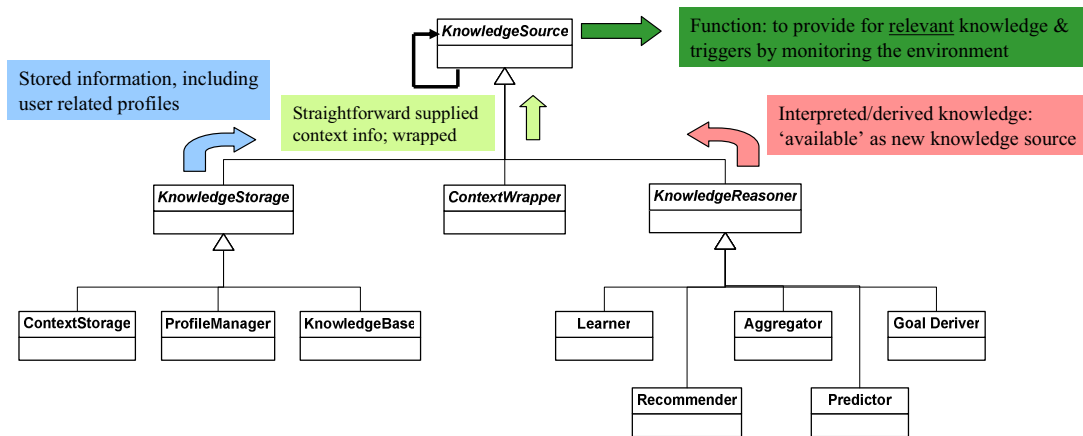


Figure 40: Types of Knowledge Source (by functionality)

2.4.3 Knowledge Sink

Knowledge Sink [6] is a component in KMF. It is used to subscribe/retrieve a context provided by Knowledge Source. There are two ways for knowledge consumer (i.e. Knowledge Sink) to retrieve knowledge from Knowledge Source. These are synchronous [6] and asynchronous methods [6].

In the synchronous case, the Knowledge Sink sends a request to Knowledge Source and waits for its reply. The reply is the lastly generated knowledge from Knowledge Source. As for the asynchronous case, the Knowledge Sink registers the knowledge that it interests with Knowledge Source. The Knowledge Source sends notifications to the Knowledge Sink when there is an update on the knowledge. This operation is done via a publish/subscribe mechanism.

We used the second approach (i.e. asynchronous approach) in our thesis work because, considering that the application often engages in interacting with the user, it is a good idea to put amount of works on to the source side rather than the application itself. Moreover, in a dynamic environment where contexts (e.g. user location) keep changing, it is optimal to utilize the second approach as the total cost could be reduced (i.e. context source can broadcast the changes to all context sinks only once, rather than each client individually polls the context source).

According to [6], Knowledge Sink provides multiple types of interface (these interfaces are used by Knowledge Source to notify context). *IContextNotification* interface provides a callback interface to be invoked when there is an update on knowledge (i.e. context) from the Knowledge Source.

Moreover, KMF API (in particular, Knowledge Sink API) provides useful methods that we can use to access the knowledge. *getContext()* returns the knowledge in an RDF document. *subscribeContext()* is used to subscribe for our desired knowledge. *notifyContext()* is a callback function used in conjunction with the *subscribeContext()* (i.e. this function will be called when knowledge mentioned in *subscribeContext()* becomes available). *unsubscribeContext()* is used to disregard our interests in the knowledge. *getContextSource()* is used to retrieve a list of context sources that are matched with the set of criteria defined in the input parameter.

subscribeContextSources() is used to subscribe with context sources that are matched with the set of criteria defined in the input parameter. *notifyContextSource()* is a callback function used in conjunction with *subscribeContextSources()* (i.e. it is called when new sources that are matched with the set of criteria becomes available). Lastly, *unsubscribeContextSource()* is used to dismiss our interest to subscribe for the Knowledge Source.

Essentially, KMF API provides an easy-to-use set of java functions for subscribing and retrieving user contexts. The methods described in this section are used when we developed a Knowledge Sink module of IA.

2.4.4 Knowledge Broker

Knowledge Broker [6] provides functionality to register Knowledge Source and facilitate Knowledge Sink to search for the source in order to obtain a particular type of knowledge. Knowledge Broker implements multiple types of interface.

According to [6], Knowledge Broker provides multiple types of interface (to be used by Knowledge Source and Knowledge Sink). In particular, the *IContextAccess* interface offers a mechanism to discover a Knowledge Source. Moreover, the interface provides methods (e.g. *getContextSource()*, *subscribeContextSources()*) to be invoked by context consumer in order to access and manipulate context sources.

In fact, while we demonstrated a context handling feature, there were Knowledge Source and Knowledge Broker already available for our demonstration purpose. However, we needed to develop Knowledge Sink as a module of IA in order to retrieve contexts published by Knowledge Source.

2.5 Security

One of important issues in distributed systems such as the one in SPICE project is a security. An example of security concern is how to prevent an unauthorized person (e.g. attacker) from breaking into a security distributed system and illegally using the resources [25]. Basically, there are two challenges in a distributed system [25]:

1. It is difficult to protect against tapping. This implies that one could assume that all messages could be overheard, recorded for replaying or modified.
2. It is difficult to certainly identify the communication partner and thus to grant a right permission.

One could handle the first challenge by deploying cryptographic methods and digital signature. The second challenge requires certification authorities and authentication protocols [25]. In this section, we will discuss about the authorization and authentication hindrance that are one of the major concerns in the mobile networks such as the one in SPICE project. Due to time limitation, the security functions are not implemented in this thesis work although the authentication and authorization technique are partially realized via user contexts.

2.5.1 Entity Authentication and Authorization

There are four requirements for entity authentication between two entities, A and B [25] (these two entities can be ,for example, service user and service provider). Firstly, A must provide evidence of its identity that B can verify. Secondly, B cannot use A's information to impersonate A to other entity, C. Thirdly, a third party cannot impersonate A to B. Lastly, these three requirements must always hold.

There are three types of evidence. Weak authentication evidence is a simple identification code such as password or PIN code. Strong authentication evidence composes of challenge and response. A correct response to the challenge verifies an identification of the entity. Zero-knowledge authentication evidence provides a proof on identification without revealing knowledge about secret to other entities.

One could classify authentication based on cryptographic methods, namely, authentication with Secret Key Cryptosystems and authentication with Public Key Cryptosystems [25]. For authentication with Secret Key Cryptosystem, an entity, A, creates a nonce (i.e. a symbol that is constructed in order to identify the present conversation), and sends it to another entity, B. B then encrypts the nonce by using a shared secret key, SK_{AB} , and sends back to A. A decrypts the nonce and compare it with the original nonce. If it is matched, A can infer that he is now communicating with B since only B knows the shared secret key and the nonce.

In fact, if A and B never communicate with one another before, they need some ways to agree on shared secret (e.g. SK_{AB}). An **authentication server** [25] is often used by A and B as a mediation for them to exchange the shared secret. The participants, A and B, and the server execute an authentication protocol such as **Needham and Schroeder's secret key authentication protocol** [35] in order to exchange authentication knowledge.

As for an authentication with Public Key Cryptosystems, there are two ways to prove identification of an entity [25]. The first way is letting one entity, A, to digitally sign a challenge that

is sent from another entity, B. The second way is letting one entity, A, to decrypt a challenge that is encrypted with A public key and sent by another entity, B.

In practice, a certificate (i.e. a structure that is exchanged among parties in order to provide proof of identity) is often used as the evidence [25]. The certificate contains: version number, identification of the owner, owner's public key, description of encryption algorithm, period of validity of the certificate, identification of certificate, description of algorithm used for generating the issuer's signature, identification of the issuer and the issuer's digital signature. An issuer of a certificate is a trusted party called **Certification Authority (CA)** [25].

In the real world, there are several issuers of certificates. The issuers form a hierarchy of trust called **trust model** [25]. There are two types of hierarchical model: a **strict hierarchical model** [25] and a **reverse certificate hierarchical model** [25]. In strict hierarchical model, the CA that is in an upper part of hierarchy certifies (i.e. issues certificate to) CA that is in a lower part of hierarchy. In reverse certificate hierarchical model, CA that is in a lower part of hierarchy can certify CA that is in an upper part of hierarchy. Hence, it is more efficient (e.g. shorter certification path) to verify certificates issued by other CA that are in the same level of the tree.

In fact, the key (i.e. evidence) exchange method in connection with entity authentication is based on **key transport technique** [25] (i.e. a technique that one party derives a secret key and securely transfers it to another party). Alternatively, **key agreement** [25] technique can be used. Key agreement is a process of key derivation in which each party contributes to part of secret. Specifically, an entity, A, computes a confidential value, Φ_A , based on its chosen secret key X_A and transmits it to another entity, B. B also computes a confidential value, Φ_B , based on its chosen secret key X_B and transmits it to A. Now, both entities have access to the shared secret Φ_C that is a function of Φ_A and Φ_B . It is worth noting that A and B do not have a knowledge on each other secret keys (i.e. A does not know X_B while B does not know X_A) but they have a full knowledge on the shared secret Φ_C . An example of key agreement protocol is **Diffie-Hellman key agreement protocol** [25].

Despite effectiveness of the key agreement protocol, the protocol suffers for a well-known **man-in-the-middle attack** [26]. An active intruder, C, can pretend to be A communicating with B and at the same time to be B communicating with A. To solve this, digital signature and encryption technique are often utilized to encapsulate the key being exchanged. An example of the protocol using this technique is **Station-to-Station key agreement protocol** [25].

Authorization property is indirectly followed from authentication property. As aforementioned, authentication provides a proof on entity identification. Authorization ensures that an identified entity has a right permission to invoke resources. If an intruder cannot impersonate the entity, authorization will not be compromised (i.e. a service user cannot access to forbidden service by pretending to be other users). However, this is based on an assumption that authorization rules of individual himself are strictly followed (i.e. a service user cannot trick a service provider that he has a permission to his forbidden service by any other means).

Due to time limitation, in this thesis, we only demonstrated how authorization can be enforced through user contexts. In our scenarios, service providers publish authorization rules through KMF. Specifically, the service provider grants or restricts user permission to the service by using two contexts, namely, authorization context and restriction context. We assumed that the authentication property has already established. That is, a user is a person he claimed to be.

Related technologies

Therefore, upon IA is notified by the authorization context of the user, IA enforces the rules accordingly. It is worth noting, however, that, in practice, the methods presented in this section are important to ensure authorization and authenticity property.

Chapter 3

Requirement Analysis and Design

In our proposed architecture presented in Figure 3, IA has been added to SCE. Basically, IA takes a service request from the end user, searching for a service that has a goal corresponding to the one inferred from the request, interacts with ACE in order to compose for a service and deliver it to the end user. Throughout its operation, IA also interacts with KMF in order to retrieve user contexts and use the contexts as a basis when selecting for service candidates.

In this chapter, functionalities of IA will be presented. Essentially, the functionalities are derived from a set of IA requirements. We conduct analysis on the requirements and set up functionalities that are essential for IA to meet all requirements. The functionalities are classified based on their operation goals. The environment that IA operates on will be discussed as it is the most important input to functionality design. Later in the chapter, we delineate an overall design and detailed design of IA for each set of functionalities.

3.1 Intelligent Agent (IA) Requirement Analysis

The IA requirements are: to demonstrate technical feasibilities and solutions and to propose ideas to improve SCE in three areas. These are:

- service composition based on natural language request
- service composition based on user context (e.g. user location)
- service restriction based on a derivation of user authentication and authorization

The first requirement indicates that IA must understand the request in natural language. Based on a meaning of the request, IA must find relevant service components and compose them together. The service compositions in SCE should be expressed in SPATEL so that the compositions can be invoked in SEE. As IA interacts with end user, it should provide an easy-to-use interface and process the request in reasonable amount of time. In other words, the first requirement indicates functionalities 1-2 and 3-11 presented below.

The second requirement indicates that IA must realize and understand user context. Based on the context, IA must produce a service composition accordingly. In other words, the second

requirement indicates functionalities 12-16. Furthermore, the first requirement and the second requirement indicate that IA should be able to process natural language request and user context at the same time. This specifies functionality 17.

The third requirement indicates that IA must realize and understand security context. Based on the context, IA must enforce security policy accordingly. In other words, the third requirement indicates functionalities 18-19.

Here is a list of properties (functionalities) that IA should have:

1. **Easy-to-use Interface:** IA should have an easy-to-use interface. The interface should take: 1) service request from the user and 2) user preference (e.g. whether the user wants to activate context handling functions) as inputs. The interface should return appropriate service composition to the user
2. **IA Performance:** The IA should be able to give an answer to a user within a reasonable amount of time. The amount of time is vary upon individual tolerance, for reference, it is defined as less than 5 minutes
3. **Information Extraction:** An ability to highlight words (i.e. keywords) that are functioned as noun, verb and adjective in a sentence (i.e. user request)
4. **Service Discovery:** An ability to search for services that have goals corresponding to keywords
5. **Service Weighting:** An ability to assign weights to discovery services. A weight denotes a likeliness of services to match user demands
6. **Service Filtering:** An ability to filter out services that have weight less than a certain threshold
7. **Service Organization:** An ability to classify services obtained from discovery process. For example, what service is executed first in the operation flow, what service is executed last in the operation flow
8. **Ranking:** Sometimes, there are more than one service composition candidate. This is an ability to rank the service composition candidates based on service semantics such as how well service composition achieve goals derived from user requests
9. **Service Parameter Extraction:** An ability to link keywords with service parameters (i.e. service input)
10. **ACE-Interface:** An ability to communicate with ACE. Ideally, IA requests ACE to generate service composition in SPATEL. The interaction between ACE and IA was supposed to be done via web service. However, as discussed in section 2.3.1, the prototype version of ACE does not support web service interface. Therefore, the interaction between ACE and IA is accomplished in source code level.
11. **NLP-Interface:** An ability to understand semantics of the user request, in natural language, via MRS and use it to improve a result of service composition

12. **KMF-Interface:** An ability to integrate with KMF. KMF is a context handling platform. It consists of three components, namely, Knowledge Source, Knowledge Sink and Knowledge Broker
13. **Context-Subscription:** an ability to subscribe with Knowledge Source. An entity that subscribes with Knowledge Source is able to retrieve user contexts (e.g. user locations)
14. **Context-Retrieval:** an ability to retrieve context from Knowledge Source
15. **Context-Understanding:** an ability to parse context in RDF that is returned by Knowledge Source
16. **Context-Handling:** an ability to select service candidate based on user contexts
17. **Synchronization:** an ability for IA to be able to process the user request and subscribing for user contexts at the same time
18. **Authentication/Authorization-Retrieval:** an ability to retrieve user privileges
19. **Authentication/Authorization-Handling:** an ability to enforce security rule when user authentication/authorization are changed

In section 3.5, we will discuss a design decision to support these functionalities. However, before the functionalities (such as functionality 3) can be realized, we need to develop configuration files to support these functionalities. This will be discussed in Section 3.4. Moreover, we will discuss an overall design of IA on section 3.3 in order to provide a big picture of this thesis.

3.2 IA environment

We have illustrated IA environment in Figure 3. IA directly interacts with ACE and KMF. The input parameters produced by IA and the SPATEL file produced by ACE (according to IA request) are used as inputs of SPATEL execution engine. As discussed in section 1.2, due to ACE limitations, IA constructs a service discovery and service composition by itself. However, IA uses ACE to export service composition into SPATEL. Moreover, IA retrieves user contexts produced by knowledge source via knowledge sink. Based on the context that IA receives, IA enforces access to services or construct service compositions, accordingly.

At the time of writing this thesis, the SPATEL execution engine is unfortunately not ready for automatic execution (we have to execute the SPATEL execution engine manually using 1) the SPATEL files and 2) the input parameter derived from user request) and the web service used for our demonstration purpose does not physically exist. Therefore, we do not focus on SPATEL execution engine in this thesis.

As shown in Figure 3, there are external files that IA uses. Among of them, service repository, goal ontology and input/output ontology are available for us. However, we have modified them to suit with our purposes. For example, we have added more services into the repository. Also,

we have added more service inputs/outputs and service goals into the corresponding ontology files. These files are shown in Appendix C.

Context ontology is also available. It is used by KMF components. As the contexts that we handle in this thesis are simple, we are not interested in context ontology. In other words, IA does not have to infer knowledge from context ontology. For example, IA does not have to understand **how** the concept such as “location” related with other concepts. IA only needs to know that **what** the “location” of the end user is at the moment.

In this thesis, we have developed/invented ontology and context files. These are recommended service rule, restriction service rule, dictionary file and synonym ontology.

As for MRS, it has been provided by NLP-tool developed by linguistic expert.

We will elaborate on IA files in section 3.4.

3.3 Overall design

IA is one of the core components in our proposed architecture in Figure 3. As shown in Figure 3, IA interacts with three elements, namely, a user, ACE and knowledge source (through knowledge sink). It generates service compositions based on a free-form request (e.g. natural language request). The information about service compositions will be sent to ACE in order to export the service composition into SPATEL as shown in Figure 41.

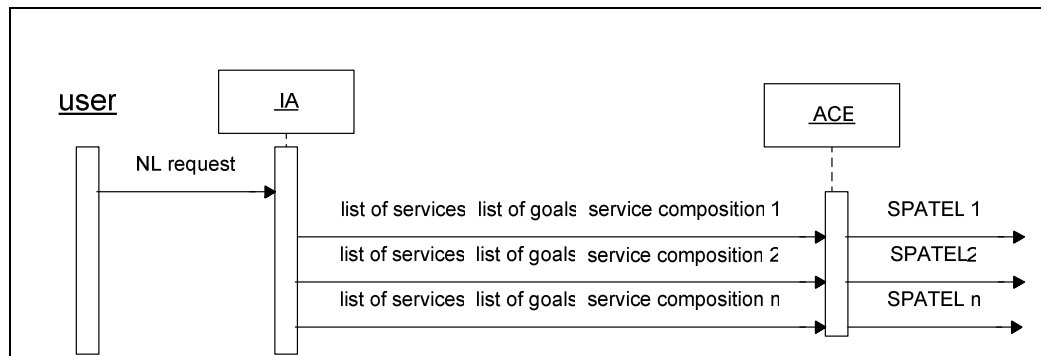


Figure 41: An interaction diagram between IA and ACE

As shown in Figure 12, IA abstractly consists of eleven components: GUI, natural language parser, information extraction component, service discovery component, service composition component, service composition sorting component, input searching component, ACE-interface component, NLP-interface component, user context parser and context-handling component. The interaction diagrams of these components are shown in Figure 42- Figure 46. The descriptions are provided below.

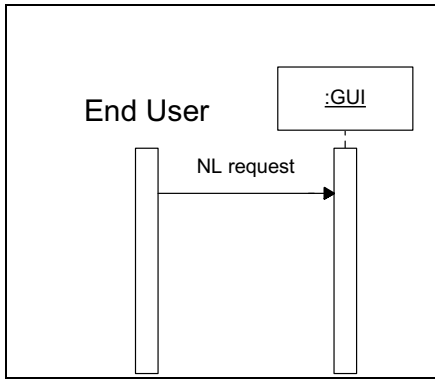


Figure 42: A sequence diagram of IA components (1)

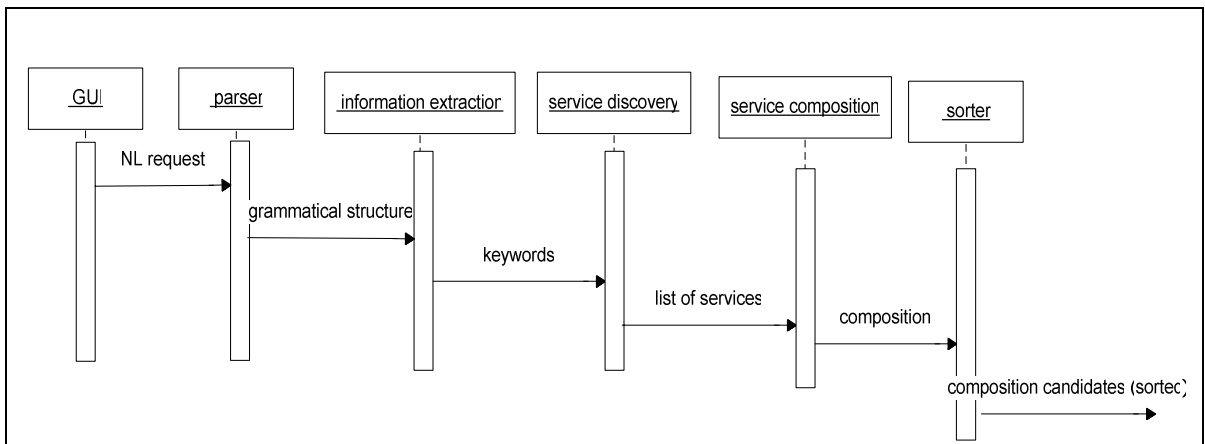


Figure 43: A sequence diagram of IA components (2)

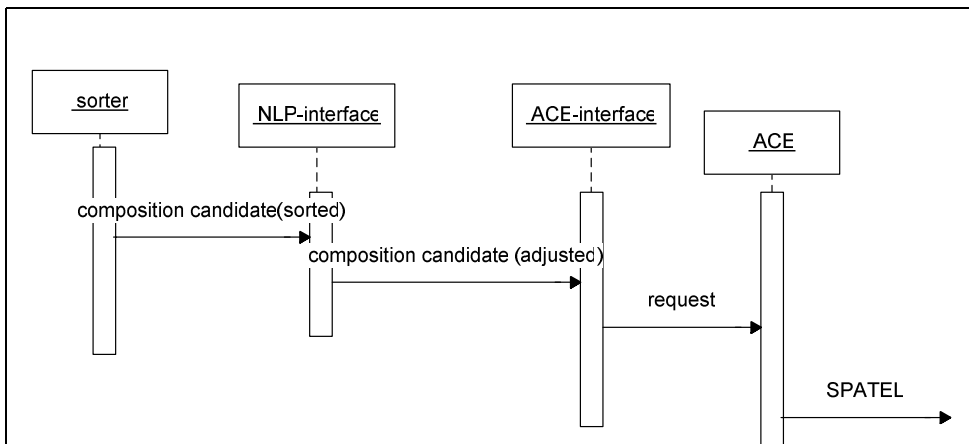


Figure 44: A sequence diagram of IA components (3)

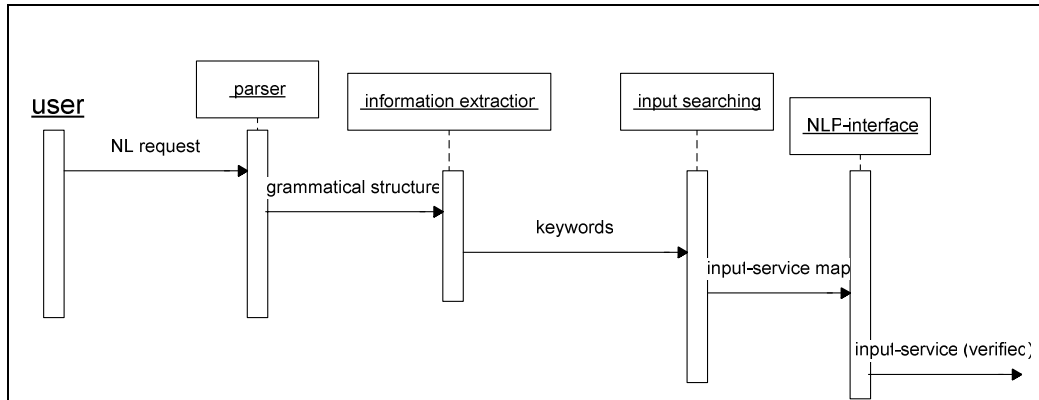


Figure 45: A sequence diagram of IA components (4)

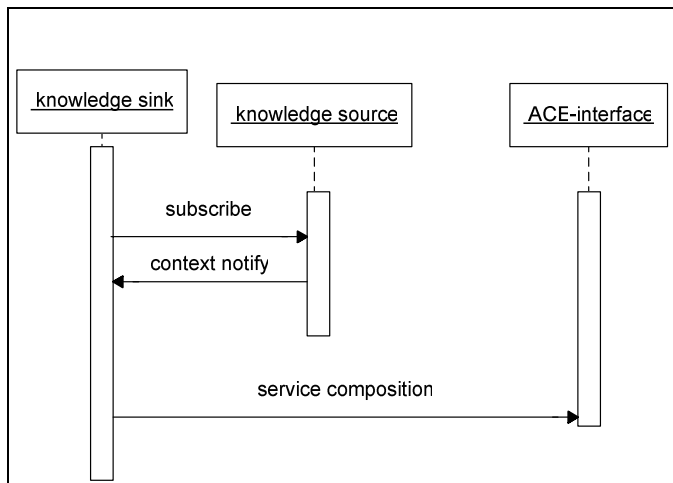


Figure 46: A sequence diagram of IA components (5)

GUI provides an interface to the end user. The end user requests a service via GUI. GUI will retrieve the request and pass the request to natural language parser. The end user can also enable/disable context handling feature and MRS feature via GUI. Also, the details about service compositions are shown to the end user via GUI.

Natural language parser works out the grammatical of sentences. It tags all words in the sentence (e.g. “translate” is a verb). Information extraction component extracts all keywords from the sentence. IA treats all noun, verb and adjective as keywords.

Service discovery component searches for all services in repository that have goals corresponding to keywords. Service composition component produces service compositions candidates from services (obtained from service discovery component) by using knowledge obtained from input/output ontology. Composite service sorting component sort service composition candidates obtained from the service composition component.

In fact, the service composition component does not provide perfect result. That is, it does not provide one precise result that satisfies user request. Rather, it provides every possible service compositions that can be derived from input/output ontology (the correct result is also one of these possibilities). For example, it may express that either the service, GetRestaurantByLocation (takes location as an input and returns restaurant address, which, in turn, is a subtype of

location as an output) or GetLocation (takes area code which is a subtype of location as an input and returns location as an output) is the service that could be executed first.

Service composition sorting component ranks the service composition based on how well the service components are group together and how well the service corresponds to the keywords. Again, the sorting does not depend on the real meaning of sentence. This implies that the sorting is not well accurate.

The NLP-tool allows the actual order of service components to be derived from a semantic form of the request in a more precise manner. Hence, the IA uses the result obtained from NLP-tool to adjust the sorted list of service composition candidates through NLP-interface component. Then, the service composition candidates are sent to ACE-interface component (the first one in the candidates is sent first). ACE-interface component interacts with ACE. Essentially, it requests ACE to export service compositions to a form of SPATEL. In the end, we will get a list of SPATEL files (the first SPATEL file is for the most appropriate service composition).

The input searching component extracts service input parameters from the sentence (i.e. user request) and assigns these input parameters to appropriate service component. For example, it extracts a word “Paris” from “get weather report in Paris” and assigns “Paris” to a service, GetWeatherReport. The input searching component depends on input/output ontology to justify type of the words (i.e. to justify that the word is an instance of service input). Again, the input searching component does get input based on a meaning of sentence. Therefore, the result of NLP-tool is used again to verify the inputs. As a result, we will get inputs of the service components from the request. The SPATEL together with inputs can be provided to SPATEL execution engine in order to generate appropriate result.

Furthermore, IA also provides context-handling functionalities through two components, namely, context parser and context-handling component. The context parser extracts context information provided by Knowledge Source residing in KMF. The context-handling component enforces rules as defined in recommendation table when user contexts are changed.

3.4 Design of IA repositories

The functionalities (i.e. properties) (such as functionality 4 and 7) presented in section 3.1 demands, among other things, a means to form concepts of service goals and service inputs/outputs which define correlation between services. One achieves this by constructing related ontologies (section 2.2) for the service goals, inputs and outputs. The approach that we used to construct ontologies is discussed in 3.4.1. Apparently, it also requires storage to keep a list of services together with its descriptions (e.g. service name, service operation) where IA can refer to. Moreover, the information extraction capability implies a demand to form concepts of vocabularies. For example, the word “find” and “search” are semantically close to each other. One achieves this by constructing synonym ontology. Next, the context-handling capability demands a mapping between service candidates and user contexts. One achieves this by establishing a table that contains exclusive rules.

The context-handling capability also demands a mapping between a user context and a list of restricted services. The mappings are kept in the restriction file. Lastly, NLP demands a way for IA to apprehend a result of NLP-tool. To achieve this, two repositories are necessary.

Firstly, a dictionary file is used to ensure compatibility between IA and NLP-tool. Secondly, an XML file is used to keep an analysis result of NLP-tool. Relationships between properties and their related repository files are shown in Table 1.

In section 3.4.2 - 3.4.9, each of these repositories will be elaborated in more details.

Properties	Repositories
4	Goal ontology, Service repository
7	Input/Output ontology
3	Synonym ontology
16	Recommended service rule, Restriction service rule
11	Dictionary file, MRS

Table 1: A relationship between properties and repository files

3.4.1 Ontology construction approach

Although IA is designed to be able to operate on varieties of ontology hierarchy, it is a good idea to exhibit a well-organized ontology structures. This makes the ontologies easy to maintain and analyze. As mentioned in section 2.2.2, a well-defined ontology should avoid class cycle. A class cycle is established when one ontology class (i.e. concept) is derived from another ontology class and, at the same time, the latter is also derived from the former. The cycle property is avoided in our ontologies. For example, a goal-ontology class, *FindRestaurantByLocation*, is derived from another goal-ontology class, *FindRestaurant*, but the reverse relationship is not true.

Next, an ontology class has the same level of generality as its sibling. For example, a goal-ontology class, *FindRestaurantByLocation*, has the same level of generality as its sibling, *FindRestaurantByAddress*. In other words, both of them satisfy the same level of goal that is to find restaurant by using some methods (i.e. *ByLocation*, *ByAddress*). An intermediate class is also used to shape the hierarchy. For example, a part of goal ontology is *Find*->*FindRestaurant*->*FindRestaurantByLocation*. The ontology class, *FindRestaurant*, is an intermediate ontology class. If the intermediate class were not used, the goal ontology structure could have been flat. For instance, ontology classes, *FindAddress* and *FindRestaurantByAddress* could have been on the same level of structure. The flat structure makes ontology difficult to analyze and process.

Another design decision is that a subclass has some properties and restrictions that superclass does not have. For example, *FindRestaurantByLocation* is more restricted in a method of getting restaurant identification than *FindRestaurant*. Furthermore, the most specific concept in an ontology is an instance of class. For example, a class instance of input/output ontology class, *CityName*, *Paris* is the most specific concepts of the class. We also utilize a concept of equivalent class (section 2.2.2) in synonym ontology. For example, synonym ontology classes, “search” and “find” are declared to be equivalence. This concept facilitates a process of analysis when one would like to search for alternative words. By this way, a related ontology concept could be deduced from equivalence property without traversing the ontology structure.

In practice, a cautious design on ontologies from the beginning is important as the ontologies (goal and input/output ontology) are deployed by several service operators. Subsequent modifications on ontologies after deployment could be troublesome.

3.4.2 Service Repository

A service repository keeps service descriptions. It is read by IA on starts up. In practice, the service repository is maintained by service providers. The service repository could be a centralized web service in which IA can access. For demonstration purpose, we define a service repository as a file.

In fact, it is not essential for the end user to know the service details. The end user simply requests for a composition of service via IA. IA keeps the information obtained from the repository in built-in data structure. This speeds up the process when IA searches for a certain service to match a user request (i.e. it is faster to read from internal memory rather than the file I/O).

A structure of service repository is shown in Figure 47. As shown in Figure 47, the repository consists of five columns. The first column from the left indicates a service name that is used to identify a certain service. As it is service identification, it is unique. The second column indicates service operation. This is similar to the service operation defined in **Web Service Definition Language (WSDL)** [44]. For example, a calculator service may have two operations, namely, *addition* and *subtraction*.

The third column indicates service goal. The service goal is an objective of service. For instance, a goal of calculator service is to do a mathematical calculation. Multiple services could share the same goal. For example, *GoogleTranslate* and *BabelFish* share the same goal that is *TranslateToEnglish*. In fact, it is so common in practice where multiple services from different operators share the same goal.

It is worth mentioning the naming convention in the repository. Each word starts with a capital letter. The following alphabets are in small letters (e.g. *Translate* in *TranslateToEnglish*). This is necessary when one wants to identify keywords from the input sentence. Essentially, only the words that are in the repository (and, thus, in related ontologies) and the words that have some connection with these words (as defined in Synonym ontology) are keywords. Only, words that are keywords will be considered when IA searches for service candidates.

The fourth column is a service input indicating an input parameter when one invokes the service. A service could have more than one input parameters. In this case, the input parameters are separated by comma. One assumption to be noted is that all service inputs are mandatory (i.e. a service requires all specified input parameters).

The fifth column is a service output indicating a result that the service provides. A service could have more than one output parameters. In this case, the output parameters are separated by comma. One assumption to be noted is that all service outputs are mandatory (i.e. a service produces all specified outputs).

In Figure 47, the service *GoogleTranslate* receives *FrenchText* as its input and returns *EnglishText* as its output in order to achieve the goal, *TranslateToEnglish*. The last column indicates a non-functional property as discussed in section 2.3.1. The non-functional property is

utilized by the ACE when it ranks the service composition candidates. As one could expect from IA functionalities, the IA does not take this parameter into account.

“Component”	“Operation”	“Goals”	“Inputs”	“Outputs”	“NF”
“GoogleTranslate”	“EnglishToNorwegian”	“TranslateToNorwegian”	“EnglishText”	“NorwegianText”	

Figure 47: A structure of Service Repository

3.4.3 Goal Ontology

Service repository describes general service properties such as service name, operation, and service goal, input/output and non-functional property. However, it does not describe how services are related. Specifically, the semantic connection between services could not be found in the repository. To fulfill this deficiency, a notion of ontology described in section 2.2 is deployed. In practice, a centralized web service could be used to provide goal ontology (e.g. in OWL). For demonstration purpose, we define goal ontology in a file.

Goal ontology, constructed using OWL, defines a semantic distance between service goals in the repository. For example, a service goal, *TranslateToEnglish* is semantically close to another service goal, *Translate*. On the other hand, a service goal, *SearchRestaurantByLocation* is semantically far from another service goal, *GetBusinessInfoByLocation*. This relationship provides crucial information when one selects service candidates to meet user requests (a service discovery process will be explained later in this chapter).

In practice, the goal ontology is maintained by service providers. It is not essential for the end user to know the ontology details. The end user simply requests for a composition of service via IA. A semantic distance between service goals is calculated by counting a number of nodes that are in the hierarchy path (of the goal graph that is formed in ontology) starting from one service goal to another service goal. Sometimes, there is more than one path starting from the first goal to the second goal. In that case, the shortest path is selected as an optimal representation of semantic resemblance of the two goals. The reason is because if the goals have *some* properties in common, they are semantically close to each other for that distance, no matter how many other different properties they contain. A real-life example is a truck and a car. Although they have many different properties such as a number of wheels, the fact that they are wheeled passenger vehicles make them automobile. Therefore, the shortest path is chosen in preference over the longer paths.

During the initialization process, IA reads the goal ontology from the ontology file and keeps the ontology in IA data structure. An example of goal ontology is shown in Figure 48.


```

</owl:Class>
<owl:Class rdf:ID="TranslateToFrench">
  <rdfs:subClassOf rdf:resource="#Translate"/>
</owl:Class>
<owl:Class rdf:ID="GetLocationByIP">
  <rdfs:subClassOf>
    <owl:Class rdf:about="#GetLocation"/>
  </rdfs:subClassOf>
</owl:Class>
<owl:Class rdf:ID="GetJoke">
  <rdfs:subClassOf rdf:resource="#Get"/>
</owl:Class>
<owl:Class rdf:ID="SearchTheaterByZIPCode">
  <rdfs:subClassOf rdf:resource="#SearchTheater"/>
</owl:Class>
<owl:Class rdf:ID="SearchTheaterByPrice">
  <rdfs:subClassOf rdf:resource="#SearchTheater"/>
</owl:Class>
<owl:Class rdf:ID="GetBusinessInfo">
  <rdfs:subClassOf rdf:resource="#GetInfo"/>
</owl:Class>
<owl:Class rdf:ID="GetHolidayDatesInGB">
  <rdfs:subClassOf rdf:resource="#GetHolidayDatesByCountry"/>
</owl:Class>
<owl:Class rdf:ID="GetPriceInfoInVCE">
  <rdfs:subClassOf rdf:resource="#GetPriceInfo"/>
</owl:Class>
<owl:Class rdf:ID="GetQuote">
  <rdfs:subClassOf rdf:resource="#Get"/>
</owl:Class>
<owl:Class rdf:ID="BookRestaurant">
  <rdfs:subClassOf rdf:resource="#Book"/>
</owl:Class>
<owl:Class rdf:ID="GetHolidayDatesByCity">
  <rdfs:subClassOf rdf:resource="#GetHolidayDates"/>
</owl:Class>
<owl:Class rdf:ID="SearchTheaterByCityName">
  <rdfs:subClassOf rdf:resource="#SearchTheater"/>
</owl:Class>

```

Figure 48: An example of Goal Ontology

As shown in Figure 48, goal ontology forms a concept of goal hierarchy. The structure is characterized to be as generic as possible. On the top of structure, there is one goal (i.e. ontology class), namely, Goals, that is a parent of all other goals. Goals that belong to the same parent (i.e. a sibling) share a common objective. For example, *GetLocationByIPAddress* and *GetLocationByCellID* share one common objective that is to *GetLocation*. Goals could have multiple parents. For example, *GetRestaurantByLocationAndType* is a child of two parents, namely, *GetRestaurantByLocation* and *GetRestaurantByType*.

As aforementioned, a semantic distance between two goals can be found by traversing from the first goal to the second goal. The calculation process is time-consuming process (although it is done only once when the IA starts up) because it requires IA to visit every node in the graph with a computation complexity of $n \times k + k \times n^n$, where n is a number of nodes in the graph and k is a number of keywords extracted from the goal ontology (recall that keyword is a word that starts with capital letter). Alternatively, a restriction could be placed on the ontology structure (however, we did not put restriction on ontology structure because we want IA to be able to process every ontology (flexibility)).

In Figure 49, the structure of goal ontology is shown. The *GetBusinessInfoByPhoneNumber* is an ontology concept that has *GetBusinessInfoTree* as its sibling and has *GetBusinessInfo* as its parent. Moreover, it has *GetInfo* as their parent of parent which in turn has *Get* as its parent. A restriction could be forced such that the topmost concept (e.g. *Get*) inhabits a general keyword, the intermediate concept (e.g. *GetBusinessInfo*) inhabits a specific keyword (together with the general keyword derived from its parent) and the profound concept inhabits an extended keyword (together with the keywords derived from its parent). In this case, goals that are semantically similar are grouped together and goals that are semantically different are placed in different boundaries. In this case, one could derive a semantic distance from a structure of ontology without traversing the real path from one goal to another goal.

However, the first approach (that we took) is more flexible in that no restrictions are placed into the ontology file and thus, it is more convenient for service developers, in practice, to deploy their services.

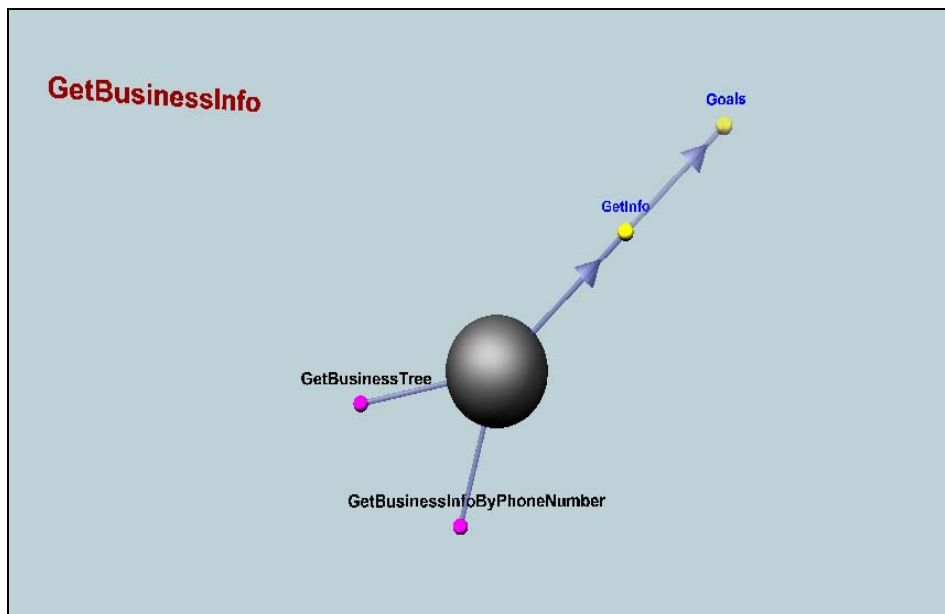


Figure 49: A hierarchy of goal ontology concept, GetBusinessInfo

3.4.4 Input/Output ontology

The input/output ontology describes a relation between service inputs/outputs. In practice, a centralized web service could be used to provide input/output ontology (e.g. in OWL). For demonstration purpose, we define input/output ontology in a file.

Practically, the input/output ontology is maintained by service providers. It is not essential for the end user to know the ontology details. The service inputs/outputs concepts are classified into ontology classes. The ontology classes form service input/output hierarchy. Two inputs are similar if they belong to the same parent. The relationship is used to organize a service composition. For example, a service, *TrafficSearch*, is executed before another service, *GoogleTranslate* because *GoogleTranslate* requires output of *TrafficSearch* as its input.

In fact, there are two types of service input/output matches, namely, exact match and analogous match. An exact match is a match that the service output of one service exactly matches with service input of another service as defined in service repository. An analogous match is a relation derived from input/output ontology.

A structure of concepts defined in input/output ontology is organized in a same way as the structure in goal ontology. Thus, it allows a flexible connection between service inputs/outputs. For example, one service input/output concepts could be derived from multiple parents. An example of input/output ontology is shown in Figure 50.

3.4.5 Synonym ontology

The synonym ontology describes a relation between keywords. Thus, keywords that have the same meaning are linked together. In practice, a centralized web service could be used to provide synonym ontology (e.g. in OWL). For demonstration purpose, we define synonym ontology in a file. The ontology is only employed by IA.

Practically, it is maintained by service developers and it is concealed from the end user. The synonym ontology is used in a process when IA searches for service goals that corresponding to certain keywords extracted from a user request. This provides more flexibility for a user to request for services. For example, a request “Search for a restaurant in Paris” has two keywords, namely, “search” and “restaurant”. If the synonym ontology was not used, the keyword “search” could not be matched with the service, “FindRestaurantByLocation”. Conversely, if the synonym ontology was used, the keyword “search” could be linked to the keyword “find” and thus, it could be matched with the service, “FindRestaurantByLocation”.

Moreover, the synonym ontology also provides more semantic relationship. The input keyword such as “food” could be linked to the keyword in associated ontology such as “restaurant” in the goal ontology. Hence, the input request could lead to a deliver of “FindRestaurantByLocation” service without a necessity of the word “restaurant” in the input sentence providing that a keyword in the input sentence properly links to the keyword in the associated ontology.

There is no restriction in the synonym ontology. Specifically, it could be defined that one keyword in the input sentence could be linked to more than one keywords in the goal ontology. In this case, both linked words are taken into account when one searches for appropriated goals matched with keywords extracted from the input sentence. It is worth noting that the synonym ontology is not used after service goal discovery process as the keywords extracted from user request have no longer been interested (i.e. after the process, the service goals will rather be used as keys in service repository for service name discovery and service organization). An example of synonym ontology is shown in Figure 51.

```

?xml version="1.0"?
<rdf:RDF
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns="http://www.owl-ontologies.com/Ontology1206625104.owl#"
  xmlns:owl="http://www.w3.org/2002/07/owl#"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema#"
  xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
  xml:base="http://www.owl-ontologies.com/Ontology1206625104.owl">
  <owl:Ontology rdf:about=""/>
  <owl:Class rdf:ID="Time">
    <rdfs:subClassOf>
      <owl:Class rdf:ID="IO"/>
    </rdfs:subClassOf>
  </owl:Class>
  <owl:Class rdf:ID="AirportInfo">
    <rdfs:subClassOf>
      <owl:Class rdf:ID="Text"/>
    </rdfs:subClassOf>
  </owl:Class>
  <owl:Class rdf:ID="NGEXStock">
    <rdfs:subClassOf>
      <owl:Class rdf:ID="StockName"/>
    </rdfs:subClassOf>
  </owl:Class>
  <owl:Class rdf:ID="LMESTock">
    <rdfs:subClassOf>
      <owl:Class rdf:about="#StockName"/>
    </rdfs:subClassOf>
  </owl:Class>
  <owl:Class rdf:ID="CityInNorthIreland">
    <rdfs:subClassOf>
      <owl:Class rdf:ID="CityName"/>
    </rdfs:subClassOf>
  </owl:Class>
  <owl:Class rdf:ID="CityInUnitedStates">
    <rdfs:subClassOf>
      <owl:Class rdf:about="#CityName"/>
    </rdfs:subClassOf>
  </owl:Class>

```

Figure 50: An example of Input/Output Ontology

```

</owl:Class>
<owl:Class rdf:ID="Reserve">
  <rdfs:subClassOf>
    <owl:Class rdf:ID="Book">
      </rdfs:subClassOf>
    </owl:Class>
  </rdfs:subClassOf>
</owl:Class>
<owl:Class rdf:ID="Examine">
  <rdfs:subClassOf>
    <owl:Class rdf:about="#Find">
      </rdfs:subClassOf>
    </owl:Class>
  </rdfs:subClassOf>
</owl:Class>
<owl:Class rdf:ID="Cinema">
  <rdfs:subClassOf rdf:resource="#Theater">
  </owl:Class>
<owl:Class rdf:ID="Get">
  <rdfs:subClassOf rdf:resource="http://www.w3.org/2002/07/owl#Thing">
  <owl:equivalentClass>
    <owl:Class rdf:about="#Find">
      </owl:equivalentClass>
  </owl:Class>
<owl:Class rdf:ID="Search">
  <owl:equivalentClass>
    <owl:Class rdf:about="#Find">
      </owl:equivalentClass>
  <rdfs:subClassOf rdf:resource="http://www.w3.org/2002/07/owl#Thing">
  </owl:Class>
<owl:Class rdf:ID="Engage">
  <rdfs:subClassOf rdf:resource="#Book">
  </owl:Class>
<owl:Class rdf:ID="Hunt">
  <rdfs:subClassOf>
    <owl:Class rdf:about="#Find">
      </rdfs:subClassOf>
    </owl:Class>
  </rdfs:subClassOf>
</owl:Class>
<owl:Class rdf:ID="film">
  <rdfs:subClassOf rdf:resource="#Theater">
  </owl:Class>
<owl:Class rdf:about="#Find">
  <owl:equivalentClass rdf:resource="#Get">
  </owl:equivalentClass>
  </owl:Class>
  
```

Figure 51: An example of synonym ontology

3.4.6 Recommended Service Rule

Recommended service rule is a repository used by IA during a process of user-context handling. It is maintained by the end user (and, hence, it is defined in a local file) so as to configure the IA to invoke a desired service composition when a certain condition (i.e. context) is satisfied. Basically, it keeps a map between service candidates (identified by a service name, service operation and input of the service) and user setting as shown in Figure 52, the rule indicates that a service candidate suitable for a user who is sitting in canteen is *SendNews*. Thus, whenever the user location (which is a type of user contexts) is changed (the IA can detect a change in user contexts by utilizing the Context Sink module which will be discussed later in this chapter), the IA checks the rule table and invokes the services defined in the table correspondingly.

The services are invoked with input parameters in the order as defined in the restriction rule (e.g. *SmsSender* is invoked with the first input parameter equals to *TrafficReport* and the second input parameter equals to a mobile number, 00475612605). In essence, there could be more than one service candidates appropriate for certain user context. In such a case, a service composition of those service candidates will be deployed. For example, if a user moves to a location, A2.05@telin.nl, a service composition of *CITYTrafficReport* (via service operation, *ReportTrafficCondition*) and *MessagingServer* (via service operation, *SmsSender*) is deployed.

A sequence of this composition is to invoke *CITYTrafficReport* first. This follows by an invocation of *MessagingServer*. A sequence of composition depends on an order of the service in the file (*CITYTrafficReport* is identified before *MessagingServer* in the file). To be flexible, a user could activate/deactivate the context handling feature via the interface of IA.

! context rule, invoke operation of service when the context occurs			
!			
! Context	Service	operation	input parameter
!			

```
"A2.05@telin.nl" "CITYTrafficReport" "ReportTrafficCondition" "Paris"
"A2.05@telin.nl" "MessagingServer" "SmsSender"
"TrafficReport,00475612605"
"Canteen@telin.nl" "GlobalNews" "SendNews"
"00475612605"
```

Figure 52: An example of recommended service rule

3.4.7 Restriction Service Rule

Restriction service rule is a repository used by IA during a process of user-context handling. It is maintained by the end user (and, hence, it is defined in a local file) so as to configure the IA to constrict an access to desired service candidates when a certain condition (i.e. context) is satisfied.

Basically, it keeps a map between service candidates (identified by a service name) and user setting as shown in Figure 53. In Figure 53, the rule indicates that when a user moves to a location, [A2.07@telin.nl](#), a service, *MailDaemon*, and a service, *EmailDelivery*, are constricted. Thus, whenever the user location (which is a type of user contexts) is changed (the IA can detect a change in user contexts by utilizing the Context Sink module which will be discussed later in this chapter), the IA checks the rule table and limit an access to the services defined in the table correspondingly. The limitation is maintained until user moves to other locations. Hence, a user who requests IA to send information when he is at [A2.07@telin.nl](#) will get the information via SMS but not via email. To be flexible, a user could activate/deactivate the context handling feature via the interface of IA.

```
! restriction rule, restrict an access to service when the context occurs
!
! Context          Service
! -----
"A2.07@telin.nl"  "MailDaemon,EmailDelivery"
```

Figure 53: An example of restriction service rule (RestrictionRule.txt)

3.4.8 Dictionary file

Dictionary file is a repository used by IA during a process of transformation a result of NLP-tool, which is in Norwegian, into English. It is maintained by IA developer so as to preserve compatibility between IA and NLP-tool. A dictionary file keeps mappings between English word and Norwegian word as shown in Figure 54.

The compatibility issue is arisen because NLP-tool handles a request in Norwegian but IA handles a request in English. For example, IA takes a user request; “find traffic in Paris and send them by email” but NLP-tool takes a user request; “finne trafikk i Paris og sende dem pr. email”.

We assumed that for simple sentences (as the ones used for our demonstration purpose in this thesis), English grammar is the same as Norwegian grammar. By this assumption (and the fact that MRS in section 2.1.5 is used), both IA and NLP-tool agree on a structure of sentence (and in fact, most of grammatical structure of the sentence does not reflect in MRS, in the first place) even though they are in different languages. In other words, a word in English sentence

(e.g. find) always refers to another word (e.g. finne) in Norwegian as defined in the dictionary file.

Hence, the word-to-word translation is sufficient to transform a result of NLP-tool into a form that IA understands. If the assumption does not hold, the one-to-one mapping between English and Norwegian words may not be possible (e.g. a number of words in Norwegian sentence and English sentence may not be equal). To be flexible, a user could configure IA to assimilate with the NLP-tool result via the IA interface.

```
! norwegian2english, dictionary file to keep a mapping between Norwegian word    ! and
English words
!
! Norwegian          English
! -----
"finne"             "find"
"trafikk"           "traffic"
"i"                 "in"
"og"                "and"
"sende"             "send"
"dem"               "them"
"pr."               "by"
```

Figure 54: An example of dictionary file (norwegian2english.txt)

3.4.9 Minimal Recursive Semantic (MRS) file

MRS [38] file stores a result of MRS analysis on a user request. The MRS analysis is performed by NLP-tool developed by linguistic expert. The file is generated by NLP-tool and it is utilized by IA.

Reading the XML file is the simplest way for IA to cooperate with NLP-tool (NLP-tool has been developed by the third party. We have no access to NLP-tool but the MRS file produced by NLP-tool). It is appropriate for our demonstration purpose in this thesis. However, this implies that the MRS analysis has to be done by NLP-tool beforehand and save it for IA to use later. In practice, the NLP-tool can be deployed as a web service where IA can query.

The result is kept in XML format. The XML contains information (e.g. a set of predicate, argument list and general quantifier) such that a relationship between services can be deduced. The XML file is shown in Figure 55. The full version of XML is not shown here in order to save space. The XML file in Figure 55 is a representation of an MRS in Figure 56 and Figure 57. As shown in Figure 56, an MRS element comprises of predicate and argument lists. In fact, the XML is directly derived from MRS. For example, <spred> is used to present predicate, <rargname> is used to present argument and so on. To illustrate a way to interpret MRS, we take some of MRS elements from MRS presented in Figure 56 and Figure 57:

```
["__finne_v_rel"<0:4>
LBL: h3 [h ROLE:ROLE]
ARG0: e2
ARG1: x5 [x WH: BOOL ROLE: ROLE PNG.NG.NUM: NUM PNG.NG.GEN: M
PNG.PERS: SECPERS]
ARG2: x4 [x ROLE: XDIM-NONINIT BOUNDED: + WH: - PNG.NG.NUM: PLUR
```

PNG.NG.GEN: N PNG.PERS: THIRDPERS]]

["addressee-rel"<0:4>
LBL: h6 [h ROLE: ROLE]
ARG0: x5]

["_pronoun_g_rel"<0:4>
LBL: h7 [h ROLE:ROLE]
ARG0: x5
RSTR: h8 [h ROLE: ROLE]
BODY: h9 [h ROLE: ROLE]]

["__trafikkforhold_n_rel"<5:22>
LBL: h10 [h ROLE:ROLE]
ARG0: x4]

["conjunction_rel"<31:33>
LBL: h21 [h ROLE:ROLE]
C-ARG: e2
L-HNDL: h24 [h ROLE:ROLE]
L-INDEX: e2
R-HNDL: h23 [h ROLE:ROLE]
R-INDEX: e22]

["coreferential_rel"<0:52>
LBL: h44 [h ROLE:ROLE]
ARG0: u45 [u ROLE: ROLE]
ARG1: x5
ARG2: x28]

The MRS elements are interpreted as follows (note that a user request is in Norwegian):

- From the first MRS element, it is elucidated that
 - finne is a verb
 - finne is a root form of a word (i.e. finn) located between position 0 and 4 of the sentence
 - finne functions as a predicate [38]
 - a variable (i.e. node), e2 is used to refer to finne (i.e. ARG0: e2)
 - the predicate, finne, takes two arguments, namely, node x5 (which is an ARG1 of the predicate) and node x4 (which is an ARG2 of the predicate). x5 and x4 are called handle (section 2.1.5)
 - ARG1 is generally the subject of a predication. For this example, it is the one to whom the user send the request. Normally, the ‘person’ of someone the user makes a request to is ‘second person’ (e.g. “you”), and this is marked in the part ‘PNG.PERS: SECPERS’ (‘PNG’ stands for ‘person-number-gender’- it is an attribute, followed by the sub-attribute ‘person’.). This is also indicated by a predicate “addressee-rel”of the second MLR-element. In a case such as ‘ROLE: ROLE’, the value of the attribute ROLE is not specified, and its value ‘role’ is then the general super-type of all roles (they could be ‘agent’, ‘receiver’, and

such). This is also the case for ‘NUM:NUM’(NUM stands for ‘number’- it could be singular or plural, and we cannot tell in this example)

- From the third MRS element, it is elucidated that
 - ‘pronoun_g_rel’ stands for a quantifier binding the variable ‘x5’- it is not one of the standard quantifiers (e.g. \forall , \exists), but serves a similar role [38]. The quantifier is called ‘generalized quantifiers’ [38]. Besides, there are other kinds of predicate that is used to identify a non-standard quantifier. One of which is a predicate_def_q
- From the fourth MRS element, it is elucidated that
 - Node x4 is corresponding to the word, trafikforhold
- From the fifth MRS element, it is elucidated that
 - The conjunction_q predicate is used to refer to the word “og” in the sentence. It conjoins two words (i.e. finn and send) together. If the word “eller” were to be in the user request, the MRS analysis would have been assigned a predicate “disjunction_rel” to it
- From the last MRS element, it is elucidated that
 - x5 is in a co-referential relation with x28 (not shown in the example), which serves as subject of “send”; so, the recipient of the first request (i.e. finn) is also recipient of the second request (i.e. send). In fact, the predicate, coreferential_rel, is also used to identify a linkage between pronoun and its subject (e.g. dem and trafikforholdene)

Initially, IA loads the data in XML into its data structures (i.e. Java Object). Based on rules defined on the predicates (e.g. coreferential_rel links a pronoun with its subject, conjunction_q links two words, the word indicated by L-Index happens before another word indicated by R-Index), IA deduces a relationship between words and goals corresponding to them.

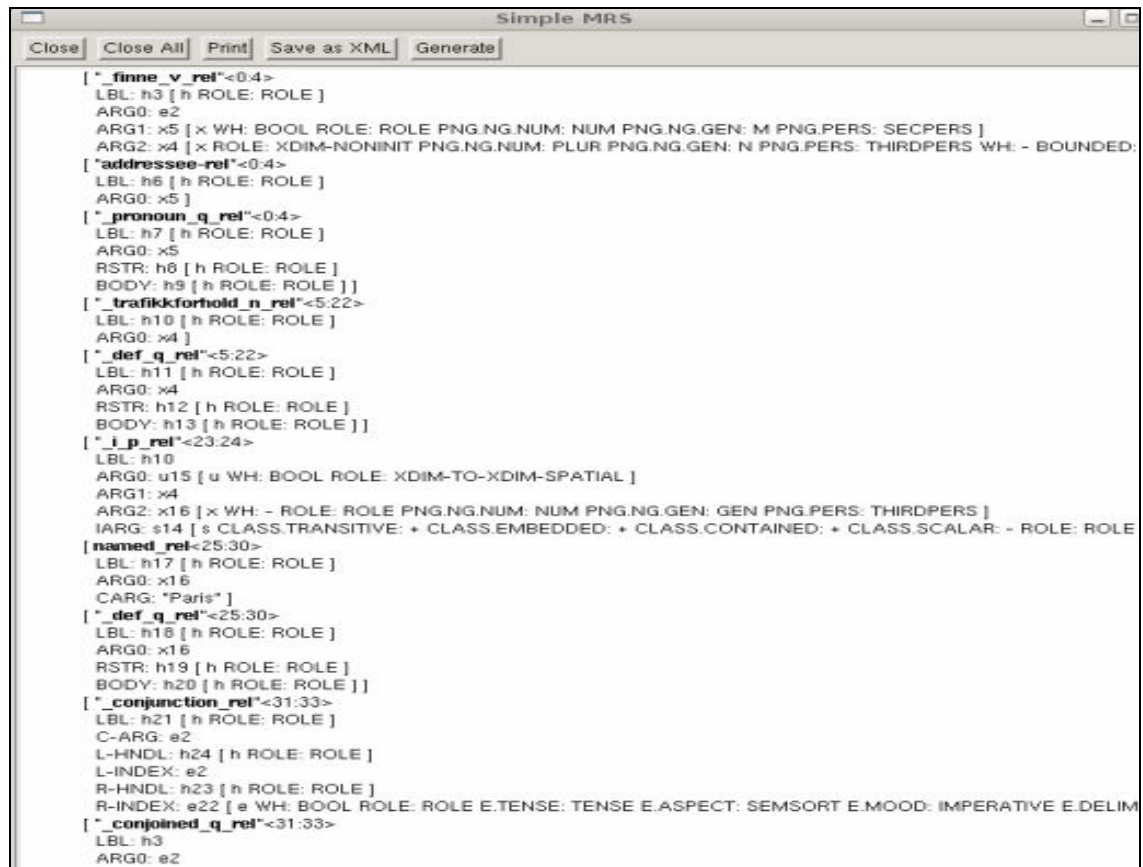
```

<mrs>
<label vid='1' /><var vid='2' />
<ep cfrom='0' cto='4'><spread_finne_v_rel /></spread><label vid='3' />
<fupair><rargname>ARG0</rargname><var vid='2' sort='e'>
<extrapair><path>SF</path><value>COMM</value></extrapair>
<extrapair><path>E.MOOD</path><value>IMPERATIVE</value></extrapair>
<extrapair><path>E.TENSE</path><value>TENSE</value></extrapair>
<extrapair><path>E.ASPECT</path><value>SENSORT</value></extrapair>
<extrapair><path>E.DELIMITED</path><value>+</value></extrapair>
<extrapair><path>PATH-TELIC</path><value>BOOL</value></extrapair>
<extrapair><path>SIT-TYPE</path><value>SENSORT</value></extrapair>
<extrapair><path>DISC-MOVE</path><value>DISCHMODE</value></extrapair>
<extrapair><path>WH</path><value>BOOL</value></extrapair>
<extrapair><path>ROLE</path><value>ROLE</value></extrapair></var></fupair>
<fupair><rargname>ARG1</rargname><var vid='5' sort='x'>
<extrapair><path>WH</path><value>BOOL</value></extrapair>
<extrapair><path>ROLE</path><value>ROLE</value></extrapair>
<extrapair><path>PNG.NG.NUM</path><value>NUM</value></extrapair>
<extrapair><path>PNG.NG.GEN</path><value>N</value></extrapair>
<extrapair><path>PNG.PERS</path><value>SECPERS</value></extrapair></var></fupair>
<fupair><rargname>ARG2</rargname><var vid='4' sort='x'>
<extrapair><path>ROLE</path><value>XDIM-NONINIT</value></extrapair>
<extrapair><path>PNG.NG.NUM</path><value>PLUR</value></extrapair>
<extrapair><path>PNG.NG.GEN</path><value>N</value></extrapair>
<extrapair><path>PNG.PERS</path><value>THIRDPERS</value></extrapair>
<extrapair><path>WH</path><value>-</value></extrapair>
<extrapair><path>BOUNDED</path><value>+</value></extrapair></var></fupair></ep>
<ep cfrom='0' cto='4'><spread_addressee_rel /></spread><label vid='6' />
<fupair><rargname>ARG0</rargname><var vid='5' sort='x'>
<extrapair><path>WH</path><value>BOOL</value></extrapair>
<extrapair><path>ROLE</path><value>ROLE</value></extrapair>
<extrapair><path>PNG.NG.NUM</path><value>NUM</value></extrapair>
<extrapair><path>PNG.NG.GEN</path><value>N</value></extrapair>
<extrapair><path>PNG.PERS</path><value>SECPERS</value></extrapair></var></fupair></ep>
<ep cfrom='0' cto='4'><spread_pronoun_q_rel /></spread><label vid='7' />
<fupair><rargname>ARG0</rargname><var vid='5' sort='x'>
<extrapair><path>WH</path><value>BOOL</value></extrapair>
<extrapair><path>ROLE</path><value>ROLE</value></extrapair>

```


Requirement Analysis and Design

Figure 55: An example of a result of MRS analysis on user request, “Finn trafikforholdene i Paris og send dem pr. email”



```
[ "_finne_v_rel"<0:4>
  LBL: h3 [ h ROLE: ROLE ]
  ARG0: e2
  ARG1: x5 [ x WH: BOOL ROLE: ROLE PNG.NG.NUM: NUM PNG.NG.GEN: M PNG.PERS: SECPERS ]
  ARG2: x4 [ x ROLE: XDIM-NONINIT PNG.NG.NUM: PLUR PNG.NG.GEN: N PNG.PERS: THIRDPERS WH: - BOUNDED: ]
[ "_addressee_rel"<0:4>
  LBL: h6 [ h ROLE: ROLE ]
  ARG0: x5 ]
[ "_pronoun_q_rel"<0:4>
  LBL: h7 [ h ROLE: ROLE ]
  ARG0: x5
  RSTR: h8 [ h ROLE: ROLE ]
  BODY: h9 [ h ROLE: ROLE ] ]
[ "_trafikkforhold_n_rel"<5:22>
  LBL: h10 [ h ROLE: ROLE ]
  ARG0: x4 ]
[ "_def_q_rel"<5:22>
  LBL: h11 [ h ROLE: ROLE ]
  ARG0: x4
  RSTR: h12 [ h ROLE: ROLE ]
  BODY: h13 [ h ROLE: ROLE ] ]
[ "_i_p_rel"<23:24>
  LBL: h10
  ARG0: u15 [ u WH: BOOL ROLE: XDIM-TO-XDIM-SPATIAL ]
  ARG1: x4
  ARG2: x16 [ x WH: - ROLE: ROLE PNG.NG.NUM: NUM PNG.NG.GEN: GEN PNG.PERS: THIRDPERS ]
  IARG: s14 [ s CLASS.TRANSITIVE: + CLASS.EMBEDDED: + CLASS.CONTAINED: + CLASS.SCALAR: - ROLE: ROLE ]
[ "named_rel"<25:30>
  LBL: h17 [ h ROLE: ROLE ]
  ARG0: x16
  CARG: "Paris" ]
[ "_def_q_rel"<25:30>
  LBL: h18 [ h ROLE: ROLE ]
  ARG0: x16
  RSTR: h19 [ h ROLE: ROLE ]
  BODY: h20 [ h ROLE: ROLE ] ]
[ "_conjunction_rel"<31:33>
  LBL: h21 [ h ROLE: ROLE ]
  C-ARG: e2
  L-HNDL: h24 [ h ROLE: ROLE ]
  L-INDEX: e2
  R-HNDL: h23 [ h ROLE: ROLE ]
  R-INDEX: e22 [ e WH: BOOL ROLE: ROLE E.TENSE: TENSE E.ASPECT: SEMSORT E.MOOD: IMPERATIVE E.DELIM ]
[ "_conjoined_q_rel"<31:33>
  LBL: h3
  ARG0: e2
```

Figure 56: MRS corresponding to user request “Finn trafikforholdene i Paris og send dem pr. email” (1)

```

Simple MRS
Close Close All Print Save as XML Generate
BODY: h26 [ h ROLE: ROLE ]
[ "_sende_v_rel"<34:38>
  LBL: h23
  ARG0: e22
  ARG1: x28 [ x PNG.NG.NUM: NUM PNG.NG.GEN: M PNG.PERS: SECPERS WH: BOOL ROLE: ROLE ]
  ARG2: x27 [ x PNG.NG.NUM: PLUR PNG.NG.GEN: N PNG.PERS: THIRDPERS WH: - ROLE: GLBTYPE49 BOUNDED: B
[ "addressee_rel"<34:38>
  LBL: h29 [ h ROLE: ROLE ]
  ARG0: x28 ]
[ "_pronoun_q_rel"<34:38>
  LBL: h30 [ h ROLE: ROLE ]
  ARG0: x28
  RSTR: h31 [ h ROLE: ROLE ]
  BODY: h32 [ h ROLE: ROLE ] ]
[ "pron_rel"<39:42>
  LBL: h33 [ h ROLE: ROLE ]
  ARG0: x27 ]
[ "_pronoun_q_rel"<39:42>
  LBL: h34 [ h ROLE: ROLE ]
  ARG0: x27
  RSTR: h35 [ h ROLE: ROLE ]
  BODY: h36 [ h ROLE: ROLE ] ]
[ "_pr_p_rel"<43:46>
  LBL: h23
  ARG0: u39 [ u WH: BOOL ROLE: LINE-TO-XDIM-SPATIAL ]
  ARG1: x27
  ARG2: x38 [ x WH: - ROLE: VIAPNT-OF-LINE PNG.NG.NUM: SING PNG.NG.GEN: M PNG.PERS: THIRDPERS ]
  IARG: s37 [ s CLASS: LINE-TO-VIAPNT ROLE: ROLE ] ]
[ "_email_n_rel"<47:52>
  LBL: h40 [ h ROLE: ROLE ]
  ARG0: x38 ]
[ "_indef_q_rel"<47:52>
  LBL: h41 [ h ROLE: ROLE ]
  ARG0: x38
  RSTR: h42 [ h ROLE: ROLE ]
  BODY: h43 [ h ROLE: ROLE ] ]
[ "coreferential_rel"<0:52>
  LBL: h44 [ h ROLE: ROLE ]
  ARG0: u45 [ u ROLE: ROLE ]
  ARG1: x5
  ARG2: x28 ]
[ "coreferential_rel"<0:52>
  LBL: h46 [ h ROLE: ROLE ]
  ARG0: u47 [ u ROLE: ROLE ]
  ARG1: x4
  ARG2: x27 ] >
HCONS: < h8 qeq h6 h12 qeq h10 h19 qeq h17 h25 qeq h21 h31 qeq h29 h35 qeq h33 h42 qeq h40 > ]

```

Figure 57: MRS corresponding to user request “Finn trafikforholdene i Paris og send dem pr. email” (2)

3.5 Design of functionalities

In this section, designs on functionalities presented in section 3.1 are given. Implementation ideas are described. The section is separated into subsections. Each subsection dedicates to each requirement.

3.5.1 Easy-to-use interface

To satisfy the first functionality, user interface capability, we develop a graphical interface using **Standard Widget Toolkit (SWT)** [31] eclipse plug-in. The graphical interface is shown in Figure 58. Basically, a user keys in a service request in natural language through the Natural Language Request box. In the bottom of the IA screen, the new button is pressed to clear out all information from the boxes. The delete button is also pressed to clear out all information from the boxes.

The difference between the new button and delete button is that the new button activate the save button. The save button activates the submit button. When the cancel button is pressed, all information from the boxes are clear and the submit button is deactivated. When the submit button is pressed, the IA analyzes the request, searches for service candidates and invoke a ser-

vice of ACE to perform service composition before deliver it to the user. The result of service composition is shown in the boxes in “Result” boundary. The “Output Dialogue” communicates information about IA operation (e.g. a result of part of speech tagging, a request that is sent to the ACE, an update of context information retrieved from KMF) to the user.

In the right hand corner, there are radio boxes to activate/deactivate user context handling functionality. Apparently, the enable box is selected to activate user context handling functionality and the disable box is selected to deactivate user context handling functionality. In other words, when the enable box is selected, the IA will instantiate the Knowledge Sink module in order to register and retrieve for an update of user context information.

In the right hand corner, there are also radio boxes to select whether the MRS will be used. If the enable box is selected, IA will verify its result with MRS. If the disable box is selected, IA will not verify its result with MRS. If MRS were to be used, the MRS (that corresponds to the user request) must have been defined in the file nlp.xml and store in the root path (in the same location as src folder).

Finally, an exit button is pressed in order to terminate IA peacefully. It is necessary to press the exit button instead of the built-in cross button at the right top-corner since the exit button will set the terminate flag that is used to notify the thread of IA (IA creates a thread to handle user contexts independently) to release its resource before completing the termination process.

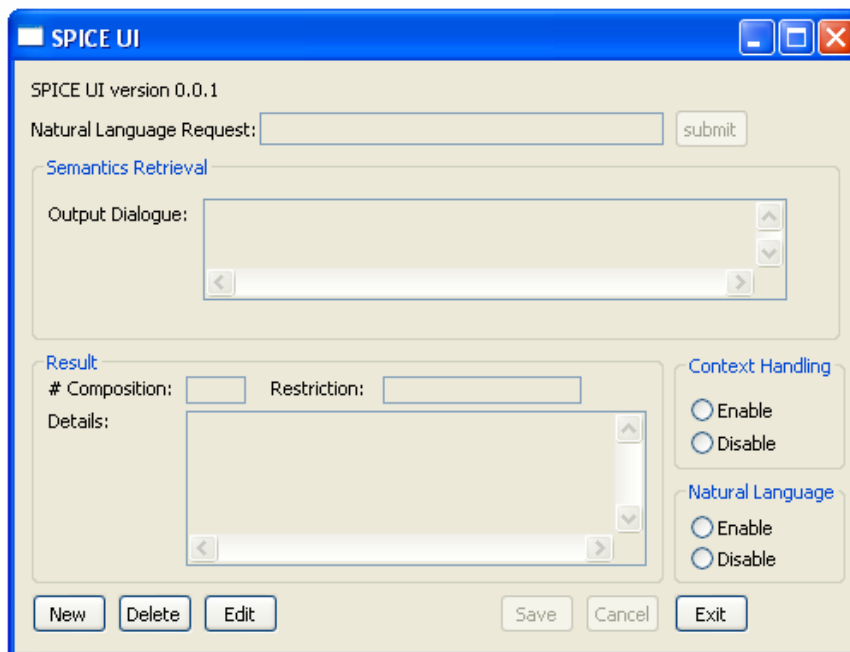


Figure 58: IA graphical interface

3.5.2 IA Performance

To satisfy the second functionality, one observes that the computation complexity depends considerably on IA service discovery process as it requires a large basic IO operation (e.g. read service repository, read ontology files etc.). Among other things, the most time-consuming operation is to read/traverse ontology such as goal ontology and input/output ontology. This is

because the IA utilizes a Jena API (section 2.2.3) to access the ontology. The size of Jena API complexity is proportional to the size of ontology structure.

Since the second functionality states that the response time of the IA should be sensible (i.e. the IA provides a real-time service to the user), the operations that are time-consuming should be done statically if possible. Based on this idea, IA statically builds up tables that keep service description and information about ontology such as a semantic distance between goals. The semantic distances between goals are used to infer a weight of keywords found in the goal. For example, a goal such as *SendSms* has two keywords, namely, *Send* and *Sms*. If the keyword, *Send*, also occurs in another goal, namely, *SendEmail*, the weight assigned to the keyword, *Send*, is equal to $100/\sigma$, where σ is equal to a semantic distance between *SendSms* and *SendEmail*.

However, if the keyword, *Send*, does not occur in any other goals, the weight assigned to the keyword is equal to 100. Hence, the keyword found in the input sentence strongly identifies the service if it is not also in other goals. If the keyword is found in more than one goal, the co-gency of identification depends on how the goals are related (i.e. semantic distance).

If the goals are not well correspond to each other (i.e. long semantic distance), the keyword is not a good service identifier as it is assigned with a small weight. If the goals are well correspond to each other (i.e. short semantic distance), the keyword is a good service identifier as it is assigned with a large weight.

From the semantic-distance table, IA constructs a weight table for every keyword found in goal ontology (notice a naming convention that a keyword starts with capital letter). During the service-discovery process, IA refers to this table in order to search for service that has goal corresponds to keywords found in user request.

Indeed, IA traverses the goal ontology only during its initialization process. During the service-discovery process, IA just searches from its internal data-structure (i.e. table). This makes its discovery process very fast. Once the corresponding goal is found, IA also searches for services that satisfy this goal through a service description table. Again, rather than reading from the external file (i.e. service repository), IA gets the information from internal data-structure.

As the time-consuming operations had been done statically, IA offers very short response time to the user although the start-up time is a long-winded (apart from the table constructions, IA also needs to load a large dictionary for part-of-speech tagging) process. Giving that the IA is loaded only once, one could claim that the approach fulfills the requirement. The only one disadvantage for this approach is that the complete information (e.g. weight of all keywords found in goal ontology) is constructed even though some of them are not used as it is not requested by the user. Furthermore, a user prediction method [32] where a user request is predicted by the application and a result is computed before the actual request could also be deployed to improve the system performance. This method will substantially improve performance of the system, especially, in a multiprocessor machine in which the prediction and an operation on user request could be performed in parallel [32].

3.5.3 Information Extraction Capability

Information Extraction Capability is an ability used to retrieve keywords from user request. It requires that IA is able to retrieve all keywords from the sentence as the keywords are important means to discover service candidates. To achieve this, IA utilizes an API from an open source community called **OpenNLP** [37]. The OpenNLP realizes an idea presented in section 2.1.1 to parse the sentence. The API performs part of speech tagging. For example, it designates the sentence “I want to get a traffic condition in Paris translated to English and I want to send it via SMS to my mobile phone” as shown in Figure 59.

In Figure 59, every word in the sentence is tagged with a grammatical element. For example “get” is tagged with VBP (verb phrase). Table 2 [33] shows a meaning of each tag in OpenNLP API.

During the information retrieval process (section 2.1.2), the word that are tagged with noun and verb are extracted. Moreover, IA also needs to consider the adjective words (those that are tagged with JJ) so that it takes the word such as “vegetarian” of “search for vegetarian restaurant” in to account. The IA highlights all these words and ignores the rest. As a result, user request is reduced to: “get traffic condition Paris English send SMS mobile phone”. There are nine words in the reduced sentence.

These words are kept in IA’s array and be compared against all goals in service repository file as discussed in subsequent sections. In fact, for a complicate request, there could be an ambiguity in the request. In this case, multiple parse trees are constructed and ranked according to the probability that they could match with the request. For simplicity, we selected the first candidate that has highest possibility to match with the request. There are also concerns about the start up time and memory usage of an application utilizing OpenNLP API. Since the application needs to load a large database of vocabularies to its internal memory on start up, the start up time could be slow. Furthermore, it requires a large memory space to keep the database. One could increase a Java memory space by providing an option to **Java Virtual Machine (JVM)** (e.g. -mx300M). Moreover, the slow start up times is also acceptable as the database needs to be loaded statically only once.

(TOP (S (S (S (NP (PRP I)) (VP (VBP want) (S (VP (TO to) (VP (VB get) (NP (DT a) (NN traffic) (NN condition)) (PP (IN in) (NP (NNP Paris)))))))))) (VP (VBD translated) (PP (TO to) (NP (NNP English)))))) (CC and) (S (NP (PRP I)) (VP (VBP want) (S (VP (TO to) (VP (VB send) (NP (PRP it)) (PP (IN via) (NP (NNP SMS))) (PP (TO to) (NP (PRP\$ my) (JJ mobile) (NN phone))))))))))

Figure 59: IA part of speech tagging

Tag	Meaning
CC	Coordinating conjunction
CD	Cardinal number
DT	Determiner
EX	Existential there
FW	Foreign word
IN	Preposition or subordinating conjunction
JJ	Adjective
JJR	Adjective, comparative

JJS	Adjective, superlative
LS	List item marker
MD	Modal
NN	Noun, singular or mass
NNS	Noun, plural
NNP	Proper noun, singular
NNPS	Proper noun, plural
PDT	Pre-determiner
POS	Possessive ending
PRP	Personal pronoun
PRP\$	Possessive pronoun (prolog version PRP-S)
RB	Adverb
RBR	Adverb, comparative
RBS	Adverb, superlative
RP	Particle
SYM	Symbol
TO	To
UH	Interjection
VB	Verb, base form
VBD	Verb, past tense
VBG	Verb, gerund or present participle
VBN	Verb, past participle
VBP	Verb, non-3 rd person singular present
VBZ	Verb, 3 rd person singular present
WDT	Wh-determiner
WP	Wh-pronoun
WP\$	Possessive wh-pronoun (prolog version WP-S)
WRB	Wh-adverb

Table 2: Word-level tag [33]

3.5.4 Service Discovery, Service Weighting and Service Filtering

Service Discovery is an ability to search for service that has goal corresponds to the keywords extracted from the user request. The reason that goal is used instead of service name is because, in reality, the service name does not always give a meaningful knowledge. A service name such as “XigniteCalendar” can be used despite the fact that the service objective is to provide financial information. IA retrieves all goals that are corresponding to some keywords. The list of goals is kept in HashMap together with their corresponding weight.

Service Weighting provides a mean for IA to deduce how well the goals correspond to the keyword. Note that HashMap gives a fast access time when one retrieve an element from the map, e.g. O(1). Service Filtering provides a mean for IA to select certain services that has goals well corresponds to the keywords. Some goals that have weight less than certain thresholds will be dropped out leaving only a list of consolidated goals.

The weight is calculated from keywords that are mapped with service goal based on some rules defined on some ontology concepts. The calculation rule is complicate and not depends only on

the number of keywords existed in the goals but also the connection between keywords and concepts of goal formed in goal ontology file. Basically, a keyword table is statically constructed by IA using goal ontology as an input. The table has two columns, namely, keyword column and a weight column.

A keyword column represents all keywords found in goal ontology. There exists a naming convention in goal ontology such that all keywords started with capital letter. A weight column represents a speciality/locality for each dedicated keyword. For example, a keyword is said to have high locality when it corresponds to a cluster of goals that are semantically close to each other in a hierarchy (as found in goal ontology). A keyword is said to have low locality if it corresponds to a cluster of goals that are far away in the hierarchy.

A keyword that is very unique (i.e. a keyword that corresponds to one and only one goal) is assigned with a weight value of 100 (i.e. 100 percent). For other keywords, the weight is calculated from dividing the maximum weight (i.e. 100) with a maximum of the distance between a pair of goals that corresponds to the same keyword.

The distances between two goals could be found by traversing from the first goal to the second goal. Sometimes, there is more than one path starting from the first goal to the second goal. In that case, the shortest path is selected as an optimal representation of semantic resemblance of the two goals. The reason is because we want to know if the keywords are used in goals that do not share many properties in common. If the goals have some properties in common, it is enough to deduce that they are semantically close to each other. Therefore, the shortest path is chosen in preference over the longer paths.

After the table is built, one could construct a list of corresponding goals based on keywords found in input sentence. In essence, a total weight (for each and every goal in the list) will be calculated from all keywords corresponding to the goal. Only the goals that have total weight more than certain thresholds will be put into the new consolidated goal list.

The certain threshold is varied among service goals. It is calculated by accumulating all weight value of all keywords in the service goal and divides the number by two. Thus, the certain threshold is a half of the maximum weight values that the service goal could have. For example, if the weight of “find”, ”restaurant”, ”by” and ”location” are 10,10,0 and 10, respectively, the maximum weight value of service goal “FindRestaurantByLocation” is 30. The certain threshold is, threshold, $30/2 = 15$. If the keywords found in the input sentence are “find” and “restaurant”, it accounts to the weight value equals to 20. As a result, the service goal “FindRestaurantByLocation” will be chosen as a candidate.

The reason we compute the threshold from the service goal itself rather than using a fixed value is because our intention is to search for every goal that corresponds to certain input keywords rather than to search for goals that consists only of very unique words. If the fixed value were to be chosen, the service goal that composes only of the general words (such as GetInfo, Get=2 Info=2) would never be selected.

In fact, we avoided selecting a restricted solution. For example, we calculated a maximum weight for each service goal by utilizing information in goal ontology. The weight is calculated programmatically rather than hardcode in the program. Therefore, if a service provider introduces a new service by updating service repository and the ontology, the new service will be recognized by IA without a modification on the code. Furthermore, if the user changed a word

in the request from “find” to “search”, IA could still deduce the same service. This is because IA defines their meaning to be equivalence. All synonym words are defined in synonym ontology file that IA consulted during the process of parsing user request. A list of consolidated goals implies a list of consolidated service as each service has a goal associated with it as defined in the service repository file. This completes a process of service discovery.

3.5.5 Service Organization and Service Ranking

Service Organization is an ability to organize a list of consolidated services. IA sorts a list of consolidated services based on an order of invocation. Input/Output ontology defines a relationship that one service provides an output to be used as an input of other services. IA organizes a list of consolidated services by consulting the input/output ontology file and service repository file.

The service repository file is enquired in order to get an input and output of the service from the service name. The input/output ontology file is enquired in order to get a similarity of input/output. After a list of consolidated service is constructed, IA assigns a relationship to every service found in the list.

Starting from taking one service as an initialized node, IA finds a service that needs to invoke before this service (e.g. service that provides its output to the input of this service). The input of one service is matched with an output of another service if one of these conditions holds:

- input of one service matches with output of another service as found in the repository file (i.e. “exact-match” relationship).
- input of one service is similar to output of another service as found in the ontology file

The exact-match relationship is assigned with a score of 100. If the second condition holds, IA will form a “general-match” relationship between two services with a score (i.e. organization score) equals to 100 divides by a distance between the input/output of the two services. The input and output are said to be similar if: 1) the superclass of service input maps to service output or vice versa and 2) the superclass of service input maps to superclass of service output.

Then, starting from the new service, IA finds a service that needs to invoke before this new service. Sometimes, there is more than one service that could provide an input for the service. In such a case, lists are created for each plausible case (with weight assigned to each case). The process is repeated until no new service can be found. As a result, a list of services is created. The first service in the list is an end node (to be sent to ACE) and the last service in the list is start node (to be sent to ACE).

The start service nodes are the services that do not require an input from output of other services. The end service nodes are the services that do not provide its output to other services. In fact, IA creates several lists by varying an initialized node (every service found in a list of consolidated service is taken as an initialized node, one at a time). After the lists are created, IA calculates a total weight for each and every list.

The weight is used to express how well the lists of service matched with user request and also how well is the organization of the list. The weight is an important component for IA to provide service ranking. The weight is calculated by accumulating organization scores as described above together with the goal weight of every goal found in the list. Indeed, the non-

functional property can also be included in the weight accumulation. Then, the IA sorts the lists by using an insertion sort algorithm.

In practice, a more efficient algorithm such as quick sort could be utilized. One at a time (list with highest score first), the IA invokes a service of ACE by providing information about service composition (e.g. a start node, end node, service list and goal list as well as order of service components (all of these are derived from the list)) to compose service in SPATEL. In the end, we can put the SPATEL generated by the ACE together with service input parameters (described in next section) to SPATEL execution engine of SEE. The SPATEL execution will invoke the service composition and produce the result. However, the process of execution of SPATEL by SPATEL execution engine has not been done in this thesis. The reasons for this are discussed in section 1.4 and 2.3.1.

3.5.6 Service Parameter Extraction

Service Parameter Extraction is an ability to extract words from a sentence (i.e. user request) and use it as inputs of service components (of service composition). To achieve this, we need a way to associate keywords extracted from the user request with service input defined in service repository.

We define ontology concepts to link between the keywords and service inputs. Specifically, the keywords are defined as an instance of service input class defined in ontology. For example, a service input, “CityInWales” is an ontology class that has “Cardiff”, “Bangor” etc. as its instances. If the service, WeatherForecastByCity, were to be selected as start service nodes, one of the keywords (“Cardiff”, “Bangor”, etc.) had to be found in the input sentence.

The keyword could match with an instance in two ways. This depends on how the instance is defined. The instance could be defined by a proper keyword (e.g. “Paris”) or it could be defined by a regular expression [10]. If the instance is defined by certain keywords, the instance will be an object of either a certain input class or one of its superclasses. If the instance is defined by regular expression (e.g. the mobile phone number), the IA will detect the occurrence of keyword by deploying regular expression rule (e.g. mobile number composes of eight digit number).

In the first case, the weight is assigned according to the class that the instance belongs to. It is proportional to the distance between the class of instance and the certain input class. For example, if the instance “Cardiff” belongs to the class “CityInWales”, the weight will be equal to 10 indicating a strong connection between keyword and service input. If the instance “Cardiff” belongs to the class “CityName”, the weight will be equal to 9 indicating a weaker connection between keyword and service input. If there are more than two keywords matched with the service input, the keyword that has higher weight will be used as the service input.

3.5.7 ACE-Interface Capability

IA interacts with ACE in order to compose for services. As discussed previously, IA is able to work out a sequence of service composition (i.e. which service operation should be executed first, which service operation should be executed last).

Ideally, the IA can transmit this information to ACE via web service (i.e. via **Simple Object Access Protocol (SOAP)** [34]) in order to generate a service composition in SPATEL. In our

work, as the ACE does not fully perform web service operation, one takes a simple approach to integrate the ACE with IA by exporting the ACE API library.

In order to use the ACE library, one needs to convert a list of objects such as an ordered list of service, an ordered list of operation in to an ACE Java Object that keeps a service composition candidate. This process takes an advantage of an encapsulation feature of object-oriented programming language and serves as a bridge between IA and ACE. Hence, a modification of internal feature of either IA or ACE does not affect the others. For example, if the SPATEL export functionality of ACE were to be changed, the changes had not affected IA.

As IA can suggest compositions to the ACE, one could reduce the work of ACE (i.e. it could skip a composition process such as to build semantic graph [3]). A result of running the IA integrated with ACE is shown in Figure 60. In this example, there are 77 compositions constructed by IA. These compositions are transformed to ACE java object and transmitted to the ACE (in our work, we invoke ACE library) in order to export (i.e. convert) them into SPATEL. As shown in **Figure 60**, the compositions have been exported and the result are kept in *spatel* folder. A sample result in SPATEL is shown in **Figure 61**.

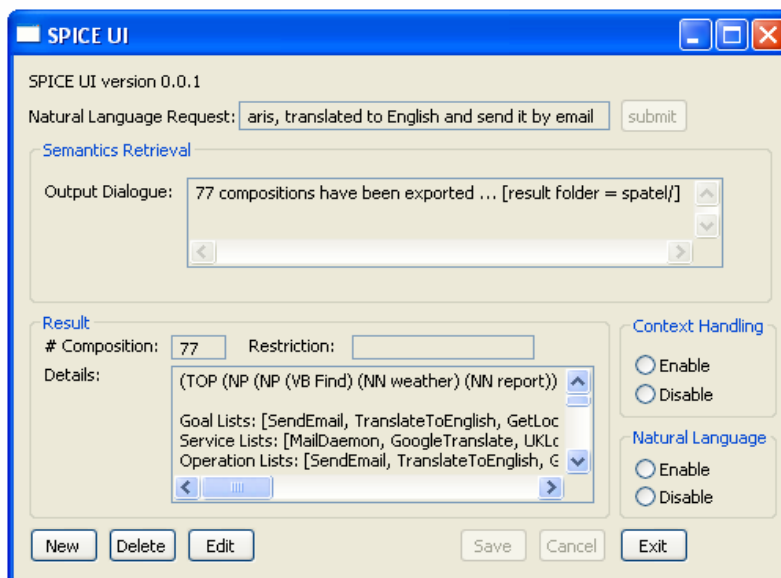


Figure 60: A result of running IA attached with ACE functionality

3.5.8 NLP-interface

An outcome, produced by IA based on a keyword/ontology-matching, is not specific enough. An intelligent NLP-tool is used to provide additional information so that appropriate compositions are given a high weight (when compare with the inappropriate compositions). Specifically, NLP-tool (we do not provide a direct description on the process of NLP-tool itself because it has been developed by external party. We did not gain an access to NLP-tool but only the result in MRS) is used to analyze a user request and produce an analysis result in the form of XML.

IA analyzes an XML generated by NLP-tool in order to get more information on the user request (and hence, service compositions). Reading the XML file is the simplest way for IA to

communicate with NLP-tool. It is appropriate for our demonstration purpose in this thesis. In practice, the NLP-tool can be deployed as a web service where IA can query. The XML is a result of conducting a natural language processing (using semantic analysis technique in section 2.1.3) on the user request. The XML contains information (e.g. a set of predicate, argument list (handle) and general quantifier) such that a relationship between services can be deduced.

The relationship determines which services should be invoked before the other services. The relationship is inferred from a sequence of events. The events are corresponding to goals that the end user wants to satisfy. IA uses keywords as indicators of events. For example, if user requests to “find traffic information in Paris and send them by email”, IA could deduce from the XML that a service that has goal, *GetTrafficCondition*, is invoked before another service that has goal, *SendEmail*. This is because the first event (that is associated with the word, “traffic”) occurs after the second event (that is associated with the word, “send”). In other words, the user wants IA to get a traffic condition and then send the traffic condition to him by email.

Therefore, if we could determine an order of words, we could deduce a sequence of events. It is worth mentioning that the order of words does not depend on a location of the words in the sentence. Rather, it depends on a meaning of conjunction and preposition that links those words together. For example, for a request such as “send traffic information in Paris to my email”, a service that has a goal, *GetTrafficCondition*, is still invoked before another service that has a goal, *SendEmail*. It does not depend on a fact that the word “send” is before the word “traffic” in the sentence.

During service composition process, IA produces a list of service composition candidates, say, l , and sorts them according to their weights. To improve a correctness of service compositions, IA assesses each composition candidate by validating it with a result from NLP-tool. If the candidate is not valid, IA will remove the candidate from its position in the list, l . Then, IA will put the candidate to the end of the list, l . If the candidate is valid, IA will continue to assess other candidates while maintaining the candidate position. By this way, the list is still ranked according to IA weighting scheme. At the same time, the candidates, those are inappropriate in the NLP-tool perspective, are degraded to the end of the list. However, among the inappropriate candidates, they are still ranked according to IA weighting scheme.

To validate a composition candidate, IA validates two conditions. Firstly, IA checks whether a number of service goals in composition is valid. Secondly, IA checks whether the composition itself is valid.

Initially, IA starts by determining a number of events (i.e. goals) from the result of NLP-tool. The number of events is equal to a number of words conjoined by a conjunction “and”. For example, a request such as “find traffic conditions in Paris and send them by email” contains two events. This is because there are two words, namely, “find” and “send”, conjoined by a conjunction, “and”. This is indicated in the XML as “and” being a predicate tag while “find” and “send” being its arguments referred by L-Index and R-Index tag, respectively. Composition candidates that are not concurred on a number of service goals (i.e. events) are considered as inappropriate. The inappropriate candidates are sifted down to the end of the list.

Next, IA determines whether the composition itself is valid. IA starts at the outermost service (a service that is invoked last), S1. IA checks whether a goal (i.e. event) of another service (a service that is invoked second-last), S2, has happened (i.e. should be satisfied) before a goal (i.e. event) of the service, S1, in the user request. The goal of service S2 (G2) is happened before a goal of service S1 (G1) if and only if: all keywords (that are corresponding to G2) in the user request are happened before all keywords (that are corresponding to G1) in the user request. This information is retrieved from the XML file. If this condition does not satisfy, the candidate is considered as inappropriate. As before, the inappropriate candidate is sifted down to the end of the list. On completion, the candidates are ranked based on both IA perspective and NLP-tool perspective. The result produced by deploying this feature is more accurate than the result based on keyword/ontology-matching alone.

As discussed in section 3.5.6, IA extracts service input parameters from the user request. Then, IA associates these input parameters with certain service components. A result obtained from NLP-tool can be used to verify that the association is indeed accurate. To verify this, IA checks whether the input instance is an input of **some keywords**, which correspond to the service goal. Keywords are representative of service goal. If the instance is an input of one of the keywords, it is sufficient to conclude that the input instance is a parameter of the service corresponding to the goal. Suppose that a user request is “find traffic condition in Paris and send them by email”. IA deduces that Paris is an input instance of the service that has a goal, *GetTrafficCondition*. Based on NLP-tool analysis, Paris is an input of traffic which is a keyword of service goal, *GetTrafficCondition*. Therefore, the deduction is verified. If the verification is not success, the connection between service and input parameter will be abandoned. As a result, an inappropriate connection of service and its parameter are avoided. However, to be flexible, end user could configure IA not to assimilate its result with the NLP-tool findings via the IA interface.

```
-<spatel:ServiceLibrary uri="http://spice.item.ntnu.no/composition_0001.spatel" xmi:version="2.0">
  -<nestedPackage name="composition_0001" xsi:type="spatel:ServicePackage">
    -<service name="composition_0001" semPattern="GIOQPE">
      -<ownedOperation name="GetTrafficCondition" xsi:type="spatel:ServiceOperation">
        <ownedParameter direction="" instanceType="string" name="" semType="IOTypes:USStreetName" xsi:type="spatel:ServiceParameter"/>
        <ownedParameter direction="" instanceType="string" name="" semType="IOTypes:TrafficReport" xsi:type="spatel:ServiceParameter"/>
        <semTag kind="goal" name="OperationGoal" semType="Goals:GetTrafficCondition" xmi:id="id"/>
      </ownedOperation>
      -<ownedOperation name="GetLocation" xsi:type="spatel:ServiceOperation">
        <ownedParameter direction="" instanceType="string" name="" semType="IOTypes:ZipCode" xsi:type="spatel:ServiceParameter"/>
        <ownedParameter direction="" instanceType="string" name="" semType="IOTypes:GeographicalLocation" xsi:type="spatel:ServiceParameter"/>
        <semTag kind="goal" name="OperationGoal" semType="Goals:GetLocation" xmi:id="id"/>
      </ownedOperation>
      -<ownedOperation name="SendSmsText" xsi:type="spatel:ServiceOperation">
        <ownedParameter direction="" instanceType="string" name="" semType="IOTypes:Text,PhoneNumber" xsi:type="spatel:ServiceParameter"/>
        <ownedParameter direction="" instanceType="string" name="" semType="IOTypes:NA" xsi:type="spatel:ServiceParameter"/>
        <semTag kind="goal" name="OperationGoal" semType="Goals:SendSmsText" xmi:id="id"/>
      </ownedOperation>
    </service>
  </nestedPackage>
</spatel:ServiceLibrary>
```

Figure 61: An example of SPATEL exported by ACE

3.5.9 KMF Interface

As discussed in section 2.4, KMF provides a means for an application in SPICE platform to retrieve user contexts. IA is attached with knowledge sink functionality that is used to access KMF. By deploying the knowledge sink module, IA registers its interest in user contexts with knowledge broker.

The knowledge broker searches for context sources that provide corresponding user contexts. IA then registers itself via the interface of knowledge source. There are two registration methods, namely, a request-response method and publish-subscribe method. In a request-response method, knowledge sink queries knowledge source for a certain knowledge and waiting for a reply. In publish-subscribe method, knowledge sink subscribes with knowledge source for its desired knowledge (i.e. context). If there is an update on the knowledge, the knowledge source will notify the knowledge sink. IA uses the second approach. This is because the second approach allows IA to concentrate on other tasks such as to satisfy the user request. IA does not have to wait for a reply and a certain event handler will be invoked when there is an update published by the knowledge source. By this second approach, IA instantly reacts to the change in user contexts (i.e. as soon as the knowledge source notifies the IA).

3.5.10 Context Subscription

As discussed in section 3.5.9, IA subscribes to knowledge source for certain user contexts. During the process, IA provides credential information and type of context to the knowledge source. This corresponds to a standard way that knowledge sink subscribes with the knowledge source [35]. The credential information is required in order to enforce a security rule. The security rule restricts an interface of knowledge source that the IA (i.e. the end user) could access. In other words, it restricts an access to a certain type of contexts that the knowledge source provides to the IA.

3.5.11 Context Retrieval

As aforementioned, there are two standard way to retrieve context information from the knowledge source. The first approach is accomplished by request-response operation. The second approach is accomplished by publish-subscribe operation. In either cases, the KMF API (described in section 2.4) provides apposite interfaces (e.g. an interface to get user context) to be invoked by a client of knowledge source.

Hence, one does not need to carry out low-level implementation (such as TCP/IP socket operation) but to call the appropriate KMF API function. Moreover, for publish-subscribe operation, one need to provide an event handler that will be invoked when the update are detected. In essence, this event handler is ensured to be invoked by KMF API and it reduces the complexity of knowledge sink developer [35].

3.5.12 Context Understanding

The user contexts are transmitted to the knowledge sink (i.e. IA) in RDF (section 2.2.1). Each element in RDF is a valid concept in context ontology. The context ontology is developed in SPICE project. It consists of a hierarchy of general concept and the more specific one (that are specific to the domain). This is in corresponds to the one proposed in [24].

IA utilizes a Jena API (section 2.2.3) to read and understand the contexts that are defined in RDF. Firstly, the IA works out the type of context received by searching for a certain property.

For example, a location context has a valid property *isLocationOf*. Secondly, the IA extracts timestamp from the context. Each context has timestamp associated with it. This timestamp is a basic element used to ensure a quality of context. A context that has timestamp older than the one received for a certain type of context should be dropped by IA. In fact, a quality of context could also be measured by other elements such as a probability of conflicts [35]. In this thesis, we only concentrate on the timestamp. Thirdly, IA extracts context value. For instance, IA pulls out a value, “kitchen”, from the node assigned in *isLocatedIn* property. Lastly, the IA checks the recommended service rule table and selects the appropriate service for certain user context (recall that IA loads the table during its initialization process).

3.5.13 Context Handling

When IA gets an update on user contexts, IA investigates the recommended service rule table and selects the appropriate service according to the user context. In fact, more than one service could be defined for a certain user context. In that case, a service composition is constructed from each corresponding service that is found in the table. This provides flexibility that the end user could define their desired service composition for certain environment (i.e. context).

3.5.14 Synchronization

Synchronization is an ability of IA to interact with end users while managing changes in user contexts. To achieve this, IA main thread creates another IA thread for context handling. While the IA main process interacts with end user (e.g. discover service, compose service etc.), the context-handling thread waits for an update from knowledge source. Therefore, IA could react to context changes while, at the same time, analyze user request. This enhances IA performance especially, in the case where multiprocessor machine is deployed. However, multithread system enlarges implementation complexity. For example, one needs to arrange a way in which the IA threads compete for the resource.

Our approach is to create a queue where a resource is exploited indirectly through the queue. For example, when the context-handling thread gets an update from knowledge source, it informs the user through the interface. In stead of writing to the interface directly, it writes the notification message to the queue. The message will be extracted from the queue later when the interface is not occupied by the IA main thread.

3.5.15 Authentication/Authorization Retrieval Capability

In this thesis, we focus on authentication/authorization property. We assume that the end-user of IA is an authenticated user (i.e. the end-user is a person that he claims to be). Furthermore, we assume that, on initialization, the end user has a full access to the list of services defined in service repository. The service operators have a privilege to limit user access right to their services by notifying IA via KMF. We also assume secure channels in KMF (i.e. an attacker cannot interfere the communication between IA and service operator). By this way, the sanctioned information is treated as one type of contexts and is retrieved by the same approach as expressed in section 3.5.11.

3.5.16 Authentication/Authorization Handling Capability

As mentioned in section 3.5.15, on start up, the end user has a full access to the list of services defined in service repository. The service operators can restrict the user access right to their services by informing IA through KMF. IA follows the same steps as in section 3.5.12 to classify user context. If the user context is an authorization context, IA will extract the context value. The context value is a service name as defined in service repository file. A subsequent request for the service will be illegitimate until the service operators withdrawn their restrictions.

Chapter 4

Implementation

In Chapter 3, we explain how IA is constructed by going through a list of IA features and design decision. In this chapter, we are going to explain IA in more detail such as what are the data structures we use to hold important elements such as ontology, service components, what are the components of IA in term of java class (as opposed to an abstract view presented in section 3.3) ,what is the IA main process and so on. Thus, Chapter 3 provides an abstract view on the IA whereas Chapter 4 provides a technical view on IA.

We start by presenting UML diagrams and classes information of the IA. Then, we discuss on data structures that UI uses. In the end of this chapter, we will discuss on IA main process. It provides details on which function calls are invoked when certain conditions are satisfied. It is worth noting that we provide an explanation on IA algorithm and IA APIs in the Appendix A of this thesis.

4.1 UML diagram and classes information

In this section, we present UML diagram of IA. The UML diagram shows association and dependence of the Java classes that are components (i.e. objects) of IA.

Implementation

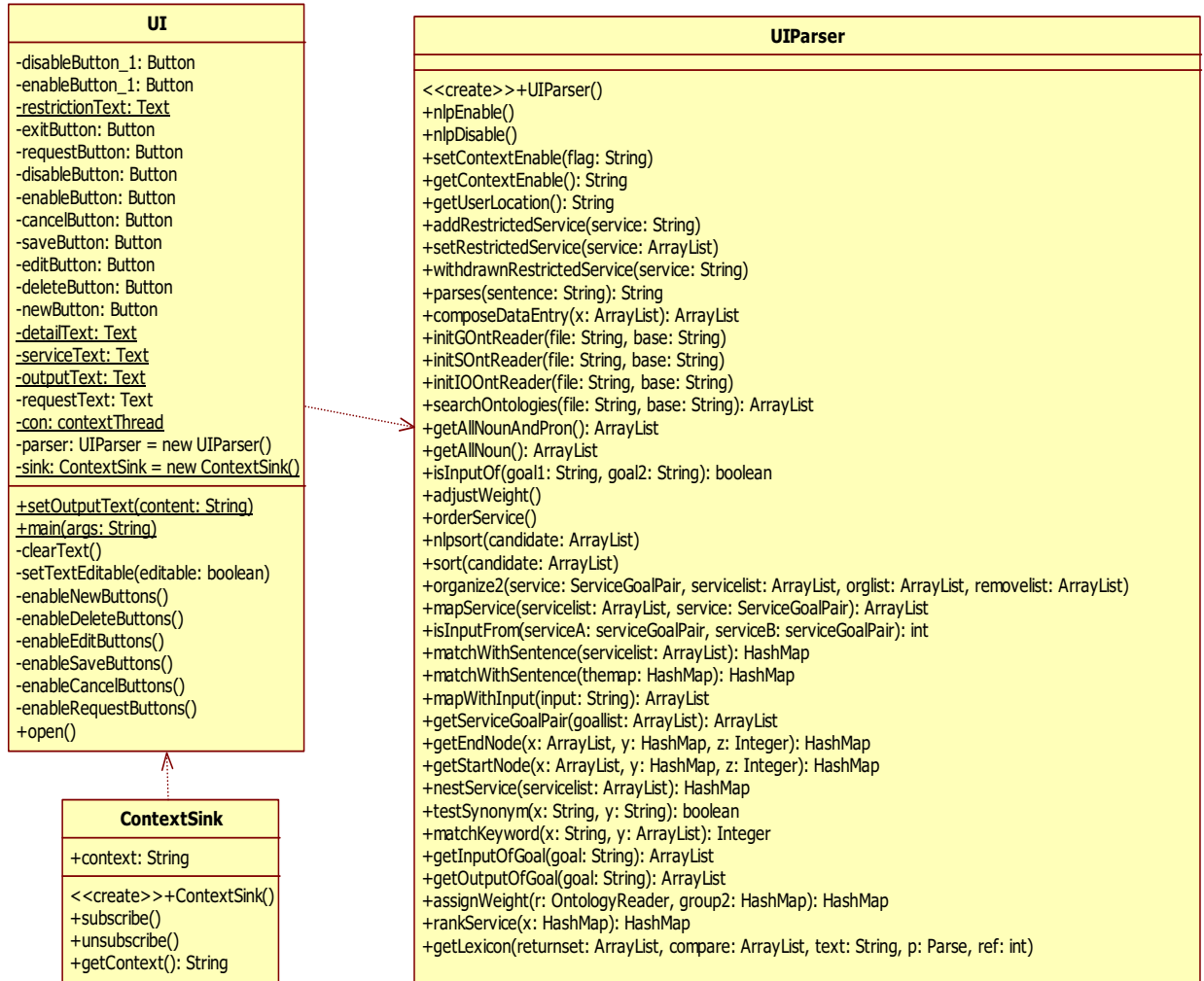


Figure 62: UML diagram of IA

In Figure 62, we have shown three components of IA, namely, UI, UIParser and ContextSink. UI is a class providing an interface to the end user. Upon it receives an end user request, it calls UIParser to parse and analyze the request. Then, UIParser will perform service discover, service composition and ranking process. UIParser also verifies the service composition it produced with MRS. The verified service composition will then be sent to ACE, in order to export it to SPATEL.

Specifically, UIParser comprises of eight abstract components discussed in section 3.3. These are natural language parser, information extraction, service discovery, service composition, service composition sorting, input searching, ACE interface and NLP interface. Furthermore, GUI and user context parser are implemented in UI class whereas context-handling component is implemented in ContextSink class.

ContextSink module notifies UI when there's an update of the context. Upon receiving the context, UI updates the screen shown to the user. Then, it invokes an operation of UIParser in order to construct a desired service composition for the end user.

Implementation

Figure 63 - Figure 67 illustrate a dependency of UIParser to other java classes. DataEntry, FlowPair, InputPair, RestrictionEntry, ServiceGoalPair, ServicePair and SC are data structure classes. DataEntry keeps a service repository's entry. It consists of six fields for storing service name, operation name, service goal, service input, service output and non-functional property. FlowPair keeps a dependency between words in the sentence. InputPair keeps a pair of two services together with the weight indicating how well the service input (of the first service) and service output (of the second service) are fitted together. RestrictionEntry stores a pair of user context and a list of services that should not be invoked when the context happens. ServiceGoalPair keeps a pair of service name and service goal (this is discussed in section 4.2.5). ServicePair stores a pair of two services that could be composed together. The difference between ServicePair and InputPair is that ServicePair does not store the matching weight. Lastly, SC stores a service composition candidate in a form that is compatible with ACE.

OntologyReader is used to parse and analyze the ontology files such as goal ontology and input/output ontology. RestrictionRuleReader is used to read the context rule files. xmlParser is used to parse and analyze the MRS. Figure 68 illustrates a dependency of xmlParser with two other classes: *predicate* and *argument*. The *predicate* and *argument* are data structure classes. The *predicate* is used to store a handle of MRS (section 2.1.5). It consists of predicate name and a list of argument (that is stored in the java object, *argument*).

A dependency between UI and three other dependent classes are shown in Figure 69. RdfReader is used by UI to parse and analyze context retrieved from ContextSink. In turn, the RuleReader (which consists of RuleEntry) is used by RdfReader to read the recommended service rule file. RdfReader invokes service composition based on the rules obtained by the RuleReader.

Implementation



Figure 63: UML diagram of UIParser and dependent classes (1)

Implementation

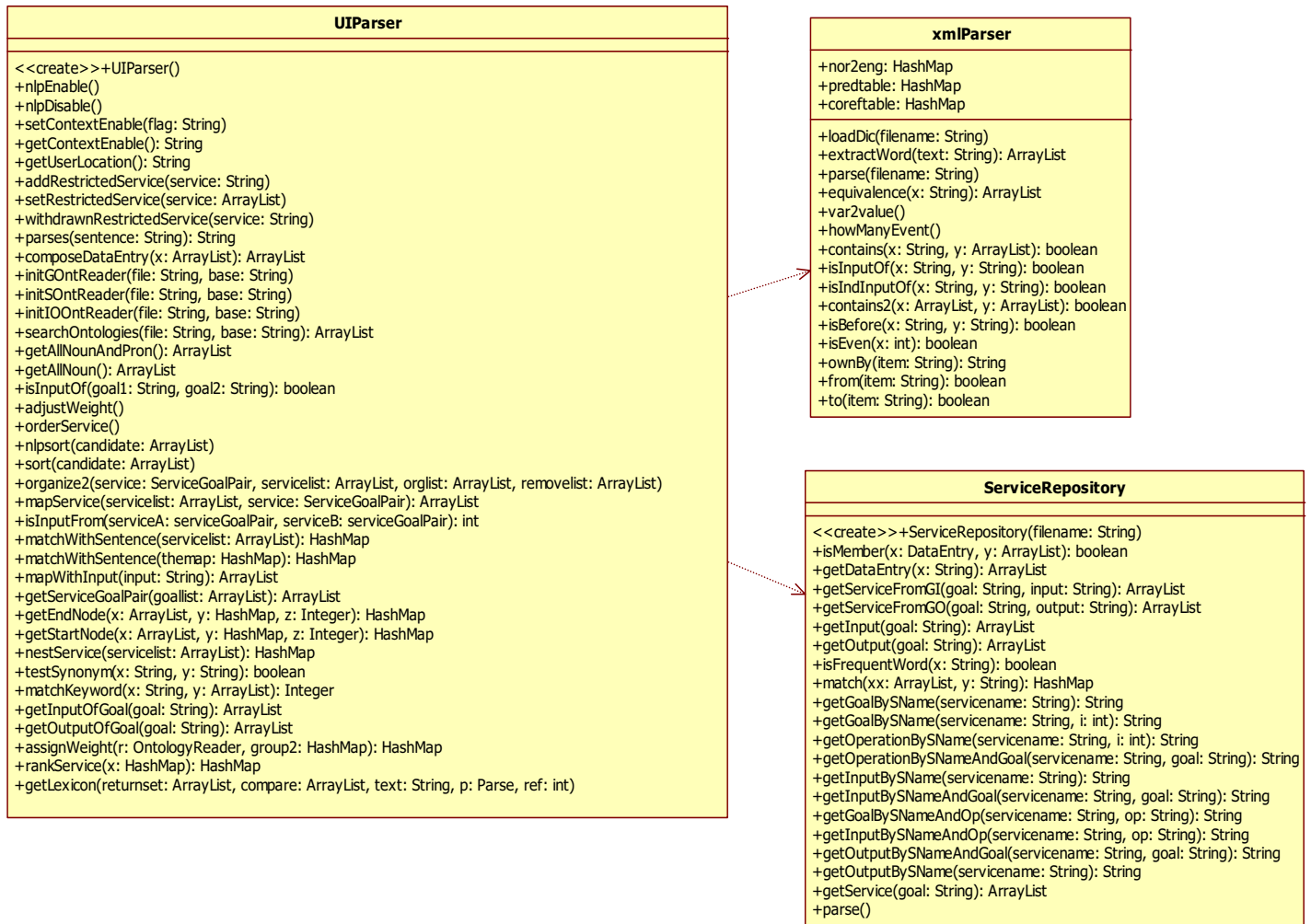


Figure 64: UML diagram of UIParser and dependent classes (2)

Implementation



Figure 65: UML diagram of UIParser and dependent classes (3)

Implementation

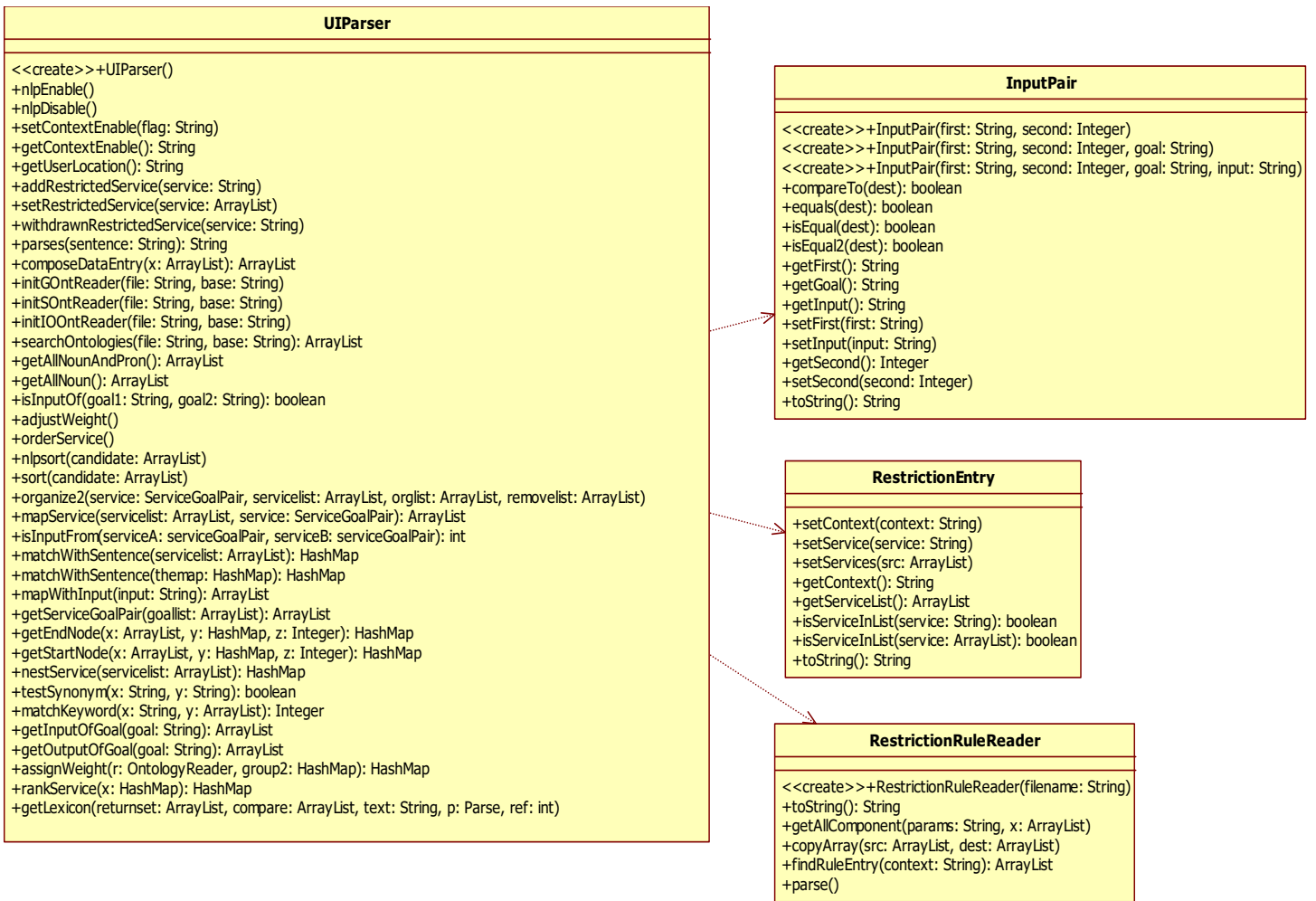


Figure 66: UML diagram of UIParser and dependent classes (4)

Implementation

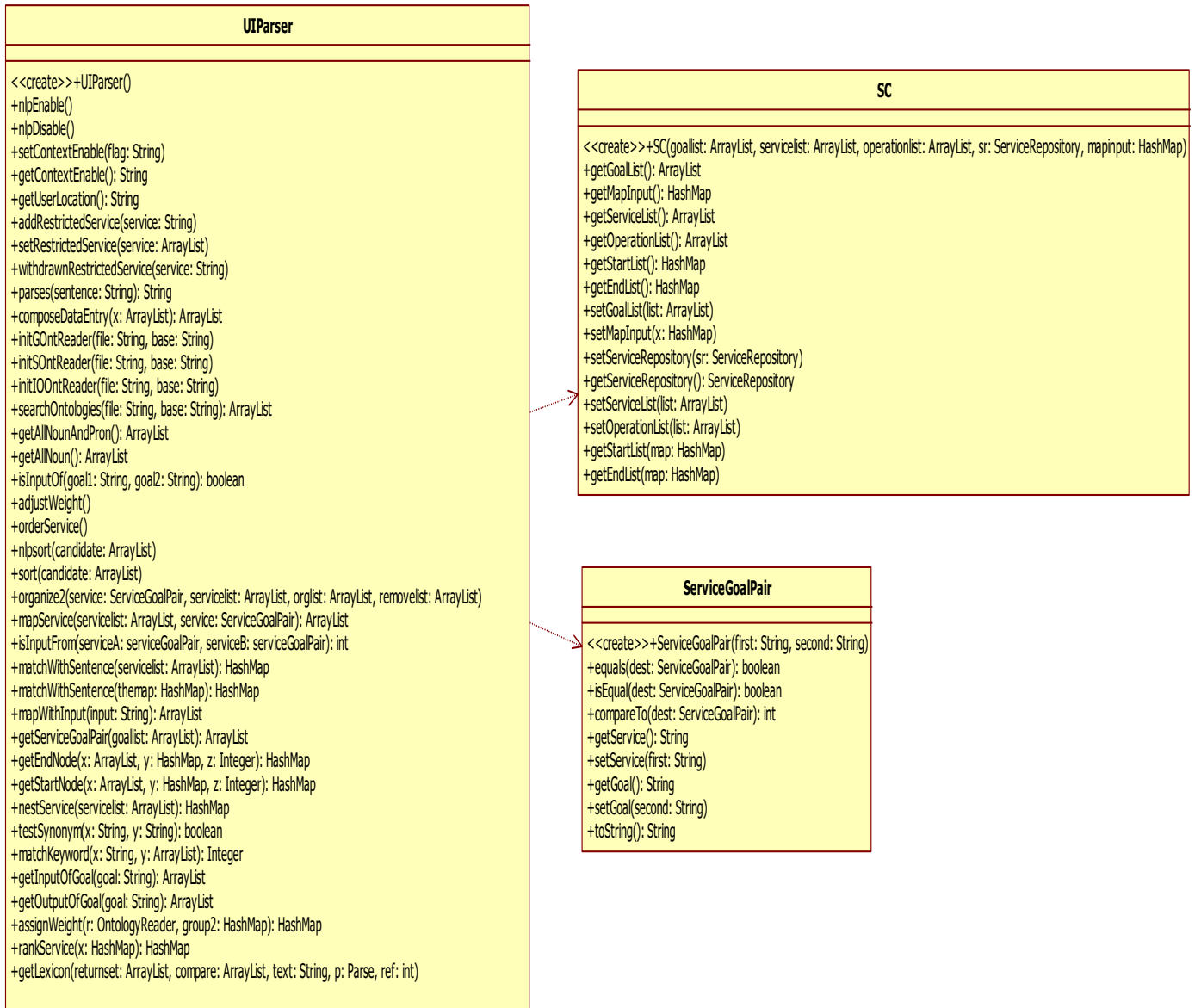


Figure 67: UML diagram of UIParser and dependent classes (5)

Implementation

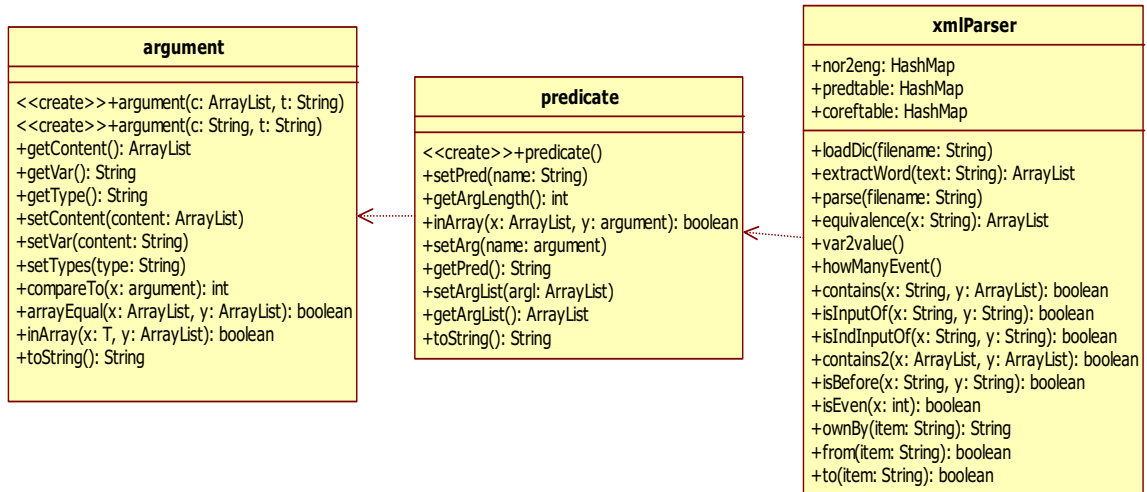


Figure 68: UML diagram of xmlParser and dependent classes

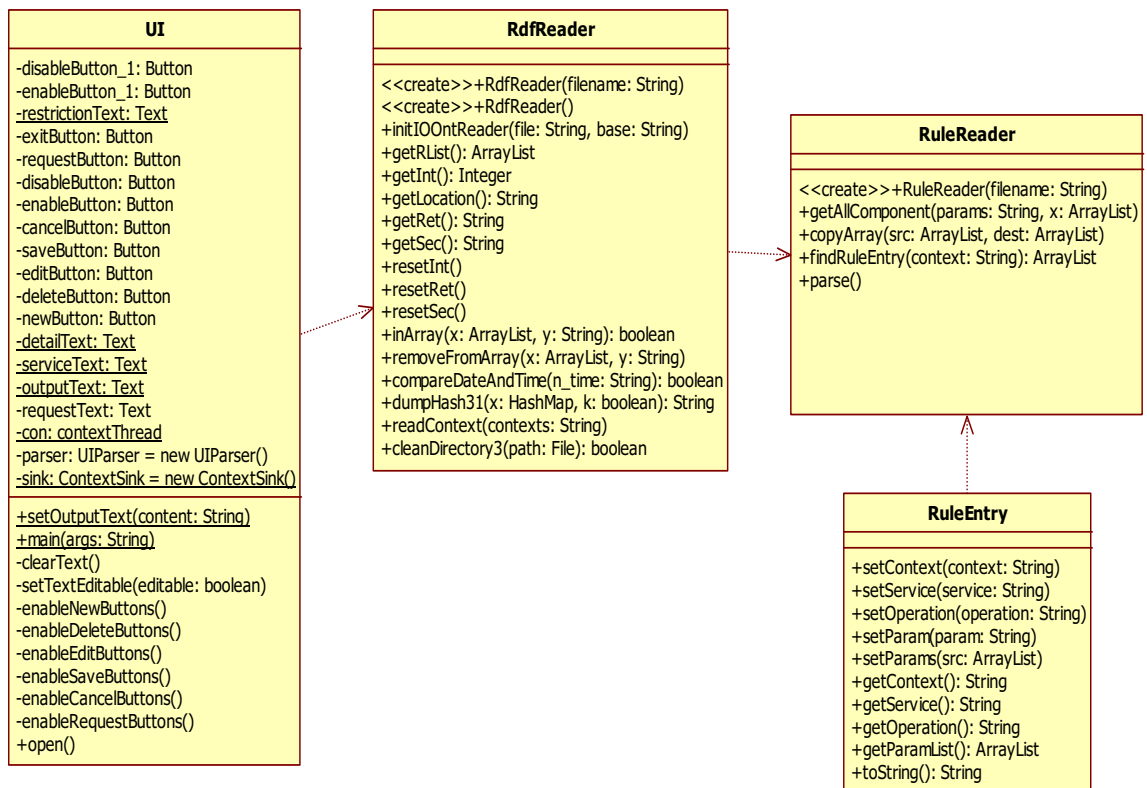


Figure 69: UML diagram of UI and dependent classes

4.2 IA Data Structure

IA stores information about service and user request in its internal data structures. One can classify data structures into two types according to their purpose. The first type of data structures is used to represent IA repositories (section 3.4). The second type of data structures is

Implementation

used to contain information that is derived by IA such as semantic distances between service goals and a weight for each keyword (section 3.5.4). In this section, we represent IA imperative data structures that belong to one of these two types.

```
ArrayList<DataEntry> rep = new ArrayList <DataEntry>();
```

Figure 70: A declaration on data structure of service repository

4.2.1 Data Structure of Service Repository

IA loads information from the service repository and store them in data structure defined in *ServiceRepository.java*. Specifically, the java object, *ArrayList*, is deployed to keep the entries extracted from the repository. *ArrayList* is preferred over a low level array (e.g. *String[]*) because it is more flexible than the latter. The size (and, thus, memory space) of *ArrayList* can be allocated dynamically at run-time while the size of array is predetermined when it first constructed [46].

Furthermore, *ArrayList* is preferred over a *Vector* as it has better performance (e.g. access time) [46]. Thus, *ArrayList* is a desired choice of data structures when one needs to keep a simple sequential of data such as a sequential of entries in the service repository. The data structure is defined as shown in Figure 70.

DataEntry is a defined java class (*DataEntry.java*) that is used to keep a service element (section 3.4.2). Intrinsically, *DataEntry* consists of service name, service operation, service goal, service input, service output and non-functional property. These components are defined in *DataEntry* class as shown in Figure 71.

```
private String component;  
private String operation;  
private String goals;  
private String inputs;  
private String outputs;  
private String nf;
```

Figure 71: A declaration on an entry in service repository

4.2.2 Data Structure of Goal, Input/Output, Synonym Ontology

IA loads information from the ontology (i.e. Goal, Input/Output or Synonym ontology) and stores them in data structure defined in *OntologyReader.java*. *OntologyReader* is a common java class that is used to process ontology models. As aforementioned, we utilize Jena API (section 2.2.3) to analyze ontology. A concern on processing ontology is a time to access and analyze concepts defined in ontology.

The processing time is proportion to a size and complexity of ontology (i.e. how the concepts defined in ontology are linked together). The processing time could be high and impractical for a very large ontology that consists of many concepts. Our solution is to cluster concepts in on-

Implementation

tology. Thus, when IA searches for a certain concept in ontology, it can examine a cluster that the concept belongs rather than examine an entire ontology domain.

For instance, if IA searches for a concept of service goal, *SendSms*, it examines a cluster of service goals that start with ‘S’ (e.g. *SendEmail*, *SendText*). By ignoring other concepts (such as *GetEmail* that starts with ‘G’), the access time to the concept is reduced. Data structures that are used to keep these clusters are shown in Figure 72.

A java `LinkedList` is used instead of `ArrayList`. Although the `ArrayList` object is more flexible (i.e. with all neat access methods available), the access time for `LinkedList` is shorter than the `ArrayList` [46]. Therefore, it is more appropriate in the situation where time complexity is a mundane.

```
private List<OntClass> ontreeA = new LinkedList<OntClass>();
private List<OntClass> ontreeB = new LinkedList<OntClass>();
private List<OntClass> ontreeC = new LinkedList<OntClass>();
private List<OntClass> ontreeD = new LinkedList<OntClass>();
private List<OntClass> ontreeE = new LinkedList<OntClass>();
private List<OntClass> ontreeF = new LinkedList<OntClass>();
private List<OntClass> ontreeG = new LinkedList<OntClass>();
private List<OntClass> ontreeH = new LinkedList<OntClass>();
private List<OntClass> ontreeI = new LinkedList<OntClass>();
private List<OntClass> ontreeJ = new LinkedList<OntClass>();
private List<OntClass> ontreeK = new LinkedList<OntClass>();
private List<OntClass> ontreeL = new LinkedList<OntClass>();
private List<OntClass> ontreeM = new LinkedList<OntClass>();
private List<OntClass> ontreeN = new LinkedList<OntClass>();
private List<OntClass> ontreeO = new LinkedList<OntClass>();
private List<OntClass> ontreeP = new LinkedList<OntClass>();
private List<OntClass> ontreeQ = new LinkedList<OntClass>();
private List<OntClass> ontreeR = new LinkedList<OntClass>();
private List<OntClass> ontreeS = new LinkedList<OntClass>();
private List<OntClass> ontreeT = new LinkedList<OntClass>();
private List<OntClass> ontreeU = new LinkedList<OntClass>();
private List<OntClass> ontreeV = new LinkedList<OntClass>();
private List<OntClass> ontreeW = new LinkedList<OntClass>();
private List<OntClass> ontreeX = new LinkedList<OntClass>();
private List<OntClass> ontreeY = new LinkedList<OntClass>();
private List<OntClass> ontreeZ = new LinkedList<OntClass>();
private List<OntClass> ontreeZZ = new LinkedList<OntClass>();

private HashMap<String,ArrayList<String>> s_table = new Hash-
Map<String,ArrayList<String>>();
```

Figure 72: Data Structures for Ontology

Ontology concepts (i.e. ontology classes) defined in goal and input/output ontology are stored in one of the lists (i.e. `ontreeA`, `ontreeB`, ... , `ontreeZ`) that is shown in Figure 72 depending on its name. However, all ontology concepts defined in synonym ontology are stored only in a list,

Implementation

ontreeZZ. This is because the number of concept (i.e. a size of) in synonym ontology created for our demonstration purpose is small.

If the synonym ontology were to be more complicate, one would also need to construct multiple lists to keep different kind of concepts as is the case for goal and input/output ontology. Lastly, a hash table is constructed to store a matching between a word that is a table's key and a list of alternative words that are table's value. The hash table is constructed by analyzing synonym ontology (i.e. ontreeZZ). One of the main reasons for selecting a HashMap as a data structure is that it provides an optimal access time. Giving a key (i.e. a word), an access time to its synonym is equal to $O(1)$ [46].

4.2.3 Data Structure of Recommended Service Rule Table

IA loads information from the recommended service rule and store them in data structure as defined in RuleReader.java. Specifically, the java object, ArrayList, is deployed to keep the entries extracted from the repository. As aforementioned, ArrayList is preferred over a low level array (e.g. String[] a) and a Vector for a simple sequential of data such as a sequential of entry in the service rule table. The data structure is defined as shown in Figure 73.

RuleEntry is a defined java class (RuleEntry.java) that is used to keep a rule element (section 3.4.6). Intrinsically, RuleEntry consists of context name (i.e. a location such as café@telin.nl), service name, service operation and a list of input parameters that are invoked when the user contexts are satisfied. These components are defined in RuleEntry class as shown in Figure 74.

```
ArrayList<RuleEntry> rulelist = new ArrayList<RuleEntry>();
```

Figure 73: Data Structure for Service Rule Table

```
private String context="";  
private String service="";  
private String operation="";  
private ArrayList<String> paramlist = new ArrayList<String>();
```

Figure 74: A declaration on an entry in Recommended Service Rule

4.2.4 Keyword table, Goal-Weight table, Keyword-Weight table and Common Goal table

Keyword table, Goal-Weight table and Candidate table are the three records utilized during a service discovery phrase of IA. The keyword table and Goal-Weight table are implementations of the abstract keyword table presented in section 3.5.4. The keyword table keeps a mapping between a service goal and a list of keywords from the user request that corresponds to the goal. The goal weight table keeps a mapping between a service goal and its semantic weight (i.e. a summation of each keyword in the goal). The keyword weight table (section 3.5.4) keeps a mapping between every keyword found in goal ontology and its semantic weight.

Implementation

The process to construct the table is discussed in section 3.5.4. The common goal table is constructed for a technical reason. It keeps a mapping between a common goal and a list of more specific goals. For example, a goal, *GetRestaurant* is a common goal for two more specific goals, namely, *GetRestaurantByLocation* and *GetRestaurantByName*. By using the common goal table, one could construct lists of consolidated goals that do not share common objective (i.e. unique). From the preceding example, one could construct two lists of consolidated goals. One comprises of *GetRestaurantByLocation* and another comprises of *GetRestaurantByName*. As IA independently performs subsequent operations on each list of consolidated goals, the two common goals are separate from each other (i.e. *GetRestaurantByLocation* and *GetRestaurantByName* are not in the same list). In other words, the IA instantly filters out an unrealistic combination of goals (a combination that consists of both *GetRestaurantByLocation* and *GetRestauranByName*). As a result, the amount of works to be done later by the service organization process (section 3.5.5) is reduced.

For each of these tables, a hash table is constructed to store the mapping. As aforementioned, one of the main reasons for selecting a HashMap as a data structure is that it provides an optimal access time providing that a table key is known [46].

```
private HashMap<String,ArrayList<String>> gktable = new Hash-
Map<String,ArrayList<String>>();
    private HashMap<String,ArrayList<String>> gctable = new Hash-
Map<String,ArrayList<String>>();
        private HashMap<String,Integer> gwtable = new HashMap<String,Integer>();
            private HashMap<String,Integer> ptable = new HashMap<String,Integer>();
```

Figure 75: Four tables that are necessary for service discovery

4.2.5 Service-Goal Pair

One of difficulties found during implementation is caused by a duplication of service names in service repository. A service name such as *GoogleTranslate* could belong to one or more entries in the repository as these entries refer to distinct operations (that satisfy different goals) of the same service. An example is shown in Figure 76. Moreover, a goal name such as *SendSms* could belong to one or more entries in the repository as these entries refer to different services that share the same goals. An example is shown in Figure 77.

Therefore, using a service name or a goal name alone may not identify an appropriate entry in the repository. To solve this, we utilize a pair of service name and service goal to identify a unique entry. This is based on an assumption that entries of the same service do not share the same goal. Generally, this is the case because entries of the same service offer different operations that satisfies different goal.

A declaration of data structure that utilizes service-goal pair is shown in Figure 78. The data structure is used to keep a mapping between an entry in the service repository and a list of pairs of keyword (taken from user request) and the degree in which the keyword is suitable to be a service input parameter. In fact, a java object, *InputPair* (defined in a source code, *InputPair.java*) is a general class object that is used to hold either a pair of input keyword and its weight as mentioned earlier or a triple of service name, a sequence weight (e.g. a degree in which an operation, *EnglishToFrench*, of service, *GoogleTranslate*, should be invoked before

Implementation

an operation, *SmsSender*, of another service, *MessagingServer*) and service goal. The second employment of *InputPair* is utilized during a process of organizing a list of service candidates. Fields of *InputPair* are shown in Figure 79.

"GoogleTranslate"	"EnglishToFrench"	"TranslateToFrench"	"EnglishText"	"FrenchText"
"GoogleTranslate"	"EnglishToNorwegian"	"TranslateToNorwegian"	"EnglishText"	"NorwegianText"
"GoogleTranslate"	"FrenchToEnglish"	"TranslateToEnglish"	"FrenchText"	"EnglishText"
"GoogleTranslate"	"FrenchToNorwegian"	"TranslateToNorwegian"	"FrenchText"	"NorwegianText"
"GoogleTranslate"	"NorwegianToEnglish"	"TranslateToEnglish"	"NorwegianText"	"EnglishText"
"GoogleTranslate"	"NorwegianToFrench"	"TranslateToFrench"	"NorwegianText"	"FrenchText"

Figure 76: Duplication of service name in service repository

"MessagingServer"	"SmsSender"	"SendSms"	"Text,SmsAddress"	"N/A"	2
"CDYNESMSNotify"	"SmsSender"	"SendSms"	"Text"	"N/A"	
"MessagingServer"	"MmsSender"	"SendMms"	"Media_Content,MmsAddress"	"N/A"	3

Figure 77: Duplication of service goal in service repository

```
HashMap<ServiceGoalPair,ArrayList<InputPair>> hash04 = new HashMap();
```

Figure 78: Data structure that utilizes Service-Goal pair

```
private String first;  
private Integer second;  
private String goal;
```

Figure 79: Fields of InputPair

4.2.6 Service Composition Candidates

As mentioned in section 3.5.7, IA invokes ACE function in order to export service composition candidates into SPATEL. The IA java object used to hold service composition candidates is defined in a java class, *SC*. The data fields (i.e. properties) of *SC* are declared as shown in Figure 80.

goallist is used to store a list of service goals for service components. *servicelist* is used to store a list of service names for service components. *operationlist* is used to store a list of service operations for service components. These lists are correlated such that a service goal, service name and service operations at the same index of *ArrayList* are referred to the same service entry. Furthermore, services in the list are properly sorted. For example, a service operation stored at the index 0 of *operationlist* is invoked after a service operation stored at the index 1 of *operationlist*, a service operation stored at the index 1 of *operationlist* is invoked after a service operation stored at the index 2 of *operationlist* and so on. On the other hand, the ACE java object used to hold service composition candidates is defined in a java class, *ServiceOperation*.

The data fields of *ServiceOperation* that are necessary for SPATEL transformation are declared as shown in Figure 81. *goals* is equivalence to *goallist* except that it keeps a list of service goals in a reverse-sort order (e.g. a service operation that corresponds to a service goal stored at the index 0 of *goals* is invoked before a service operation that corresponds to a service goal stored at the index 1). *inputs* and *outputs* keep a list of java object, *Parameter*. The data fields

Implementation

of *Parameter* are declared as shown in Figure 82. Basically, *Parameter* contains information necessary for ACE to infer service composition graph [3] such as semantic types (e.g. how well the parameter (and also the service that provides this parameter) could be used as an input of a service node) and direction of the Parameter (e.g. it provides an information about a direction of an edge of a graph, whether the edge is leading “to or from” the parameter node, whether the parameter node is an input of one service or an output of another service) and information necessary for ACE to indicate service input/output parameters.

Similar to *goals*, *inputs* and *outputs* keep a list of service parameter in a reverse-sort order. As IA and ACE employ distinct java objects (i.e. data structures) to hold service composition candidates, IA needs to convert its presentation (e.g. SC) into ACE presentation (e.g. ServiceOperation) before invoking an ACE SPATEL transformation method. The function, *getServiceOperation()*, is used to convert the IA data structure into ACE data structure so as to ensure a compatibility between the two. The conversion process is not complicated as the information required in ACE data structure can be obtained directly from IA data structure.

```
private ArrayList<String> goallist = new ArrayList<String>();
private ArrayList<String> servicelist = new ArrayList<String>();
private ArrayList<String> operationlist = new ArrayList<String>();
private ServiceRepository rep;
```

Figure 80: A declaration of SC fields

```
private List<String> goals;
private List<Parameter> inputs;
private List<Parameter> outputs;
```

Figure 81: A declaration of ServiceOperation fields

```
private static final long serialVersionUID = 1L;
private final String direction;
private final String name;
private final String semType;
private final String xmiId;
```

Figure 82: A declaration of Parameter fields

4.2.7 Data Structures for manipulating NLP result

In this section, we present data structures that store necessary information when IA converts an XML document, produced by NLP-tool, into Java objects. The information contained in XML document are loaded into IA internal memory during IA startup process (if IA is configured to utilize NLP-tool analysis result) in order to increase access speed. The data structures are shown in Figure 83.

nor2eng keeps a mapping between a Norwegian word and an English word. The mappings are loaded from a dictionary file, *norwegian2english.txt*. The data structure, *nor2eng*, is important as it helps us to maintain compatibility between IA and NLP-tool. The compatibility issue is arisen because NLP-tool handles a request in Norwegian but the IA handles a request in English. For example, IA takes a user request; “find traffic in Paris and send them by email” but NLP-tool takes a user request; “finne trafikk i Paris og sende dem pr. email”.

Implementation

We assume that for simple sentences (as the ones used for our demonstration purpose in the thesis), English grammar is the same as Norwegian grammar. By this assumption, both IA and NLP-tool agree on a structure of sentence even though they are in different languages. In other words, a word in English sentence (e.g. find) always refers to another word (e.g. finne) in Norwegian as defined in the dictionary file. Hence, the word-to-word translation is sufficient to transform a result of NLP-tool into a form that IA understands.

If the assumption does not hold, the one-to-one mapping between English and Norwegian words are not possible (e.g. a number of words in Norwegian and English may not be equal). As a result, the result cannot be utilized by IA. In practice, this assumption should not be made as it puts too much restriction on coordination between IA and NLP-tool.

Either IA or NLP-tool should be modified to support language of the others. *var2value* keeps a mapping between a variable defined in the XML file and a word refers to that variable. This mapping information is obtained from the XML file by reading an appropriate XML tag. *predtable* keeps a mapping between a predicate name and the predicate (i.e. handle) itself. The predicate is defined as a Java object. The predicate object has data fields as shown in Figure 84.

predname keeps a name of the predicate (this is same as a key of *predtable*). *arglist* is a list of argument of the predicate. The argument contains argument name and argument type. The argument name is a name of another predicate that is taken as an argument of the first predicate (i.e. *predname*). The argument type indicates a type of argument as defined in the XML file (e.g. ARG0, ARG1, L-Index, R-Index). The information about predicate and its argument list are also derived from the XML file by reading an appropriate XML tag.

Lastly, *coreftable* keeps connections between one variable and other variables. Sometimes, NLP-tool refers to the same word by using different variables (e.g. x1 and x2 are both referred to the word “sende”). The coreferential tag is used to indicate this information. IA extracts this information from the XML file and store into the *coreftable*. Hence, giving a variable, a1, *coreftable* returns other variables that refer to the same word as a1.

```
// norwegian to english
HashMap<String,String> nor2eng = new HashMap<String,String>();
// var to value
HashMap<String,String> var2value = new HashMap<String,String>();
// predicate table
HashMap<String,predicate> predtable = new HashMap<String,predicate>();
// coreferential table
HashMap<String,ArrayList<String>> coreftable = new HashMap<String,ArrayList<String>>();
```

Figure 83: Data Structure for manipulating NLP result

```
String predname = "";
ArrayList<argument> arglist = new ArrayList<argument>();
```

Figure 84: Field declarations of predicate object

4.3 IA main process

IA is an event-driven application. IA conducts its main operations based on a trigger from the end-user. In this section, we will discuss on IA main process. Initially, IA loads its GUI and

Implementation

waiting for an input (i.e. event) from the end user. Apart from loading GUI, IA also initializes the following Java objects:

- `UIParser` is created. The `UIParser` is a vital object. It is used to operate on user request and conduct appropriate service compositions for the end user
- `ContextSink` is created. The `ContextSink` is used to subscribe for user contexts. It is also used to retrieve the user context and assign the context (that is in RDF) into context variable
- `contextThread` is created. The `contextThread` is IA thread that is used to periodically check the context variable. If there is an update on the context, it will enforce the context rules accordingly (e.g. to restrict an access to a certain type of service)

Depending on an input of the end user, appropriated function calls are invoked as follows:

- If user submits a request, IA will invoke a function `UIParser.parse()` (section A.7.3) to parse the user request. Then, it will invoke functions, `UIParser.getLexicon()` (section A.7.3) to extract keywords from the user request, `UIParser.getGoalsIO()` (section 3.5.4) to extract a list of service goals from the keywords, `UIParser.createList()` to construct a common goal table (section 4.2.4), `UIParser.assignWeight()` to construct a Goal-Weight table (section 4.2.4), `UIParser.orderService()` to filter out services that have goal weight less than a certain threshold (section 3.5.5), `UIParser.organize2()` to organize the list of remaining services based on a relationship between service input and output (section A.5), `UIParser.matchWithSentence()` to associate input instances (e.g. Paris) from a user request to certain services (section 3.5.6), `UIParser.sort()` to sort a list of service composition candidates (section 3.5.5), `SC.SC()` to convert a service composition candidate (that is stored in IA data structures) into a format that is compatible with ACE (section 3.5.7), and `Test.export2Spatel()` to export the service composition candidates into SPATEL (section 3.5.7). The service compositions in SPATEL are ready to be executed by SPATEL execution engine
- If the end user would like to improve the performance of IA with a result of NLP-tool (i.e. the end user selects the “enable” radio box), IA will set up NLP Boolean flag of `UIParser` to true. When user submits a request, IA will go through the process as described in previous paragraph. However, this time, the result of IA will be verified with MRS. Specifically, there are two main function call to be invoked (these two functions are invoked only when the NLP Boolean flag is “true”). These are `xmlParser.parse()` and `UIParser.nlpsort()`. The `xmlParser.parse()` is used to parse and produce a set of predicates from the MRS (that is in XML format) created by NLP-tool (section A.7.5). The `UIParser.nlpsort()` is used to sort the list of service composition candidates (that is already sorted once by IA) by using the set of predicates (section 3.5.8). The adjusted service composition candidates will be exported to SPATEL using the same process as described in previous paragraph.
- If the end user enables context-handling functionality of IA (i.e. the end user selects the “enable” radio box), IA will invoke a function `ContextSink.subscribe()` to start a subscription on user contexts (section A.7.4). IA will also invoke a function `UI-`

Implementation

Parser.setContextEnable() to set a context enable flag of *UIParser* to “true”. If there is an update on context, the context-handling thread of IA will invoke a function, *RdfReader.readContext()* to parse user context, to check the context’s timestamp in order to ensure that it is indeed an update (i.e. it is not an old context that is just arrived because of network latency) (section 3.5.12), to check the context’s owner in order to ensure that the context belongs to the end user and to compose for a service as defined in Recommended Service Rule (section 3.4.6). Moreover, IA invokes *UIParser.setUserLocation()* (if the context is location context) to set up a new location of the end user (section 3.5.13) and *UIParser.setRestrictedService()* (if the context is authentication context) to set up a permission on the service access (i.e. to add or remove a service from restriction list)(section 3.5.16). In fact, when *UIParser* processes a user request, it will take into account the user context (if the context enable flag is “true”). In other words, it will check a current user location. It will not select services those are inappropriate to current user location as defined in Restriction Service Rule (section 3.4.7). The *UIParser* will also not select the services those are in restriction list

- If the end user disable context-handling functionality of IA (i.e. the end user selects the “disable” radio box), IA will invoke a function *ContextSink.unsubscribe()* to stop a subscription on user contexts (section A.7.4). IA will also invoke a function *UIParser.setContextDisable()* to set a context enable flag of *UIParser* to “false”
- If the end user does not supplement IA’s knowledge with a result of NLP-tool (i.e. the end user selects the “disable” radio box), IA will set up NLP Boolean flag of *UIParser* to “false”. As the NLP Boolean flag is set to “false”, the *UIParser* will not take MRS of NLP-tool into account

Hence, IA waits for a request from the end user. Once IA receives the request, IA will invoke appropriate function calls. After that, IA will wait for the request again. This process will continue until the end user pushes the “exit” button. If the “exit” button is pushed, IA will terminate its process peacefully (i.e. clear internal data structures and terminate all created threads).

Chapter 5

Result and Discussion

In this chapter, the results of running IA are presented. We have set up three scenarios to test the IA features that were mentioned in Chapter 3. The three scenarios have been presented in section 1.1 as follows:

Scenario 1

A customer, who is a vegetarian, plans to have a dinner. He is searching for a restaurant. He prefers going to the restaurant located near his house. Moreover, he wants a traffic report while he is driving to the restaurant so that he can avoid traffic jam. As a good planner, he also wants to check what the weather is going to be like before he leaves from the restaurant.

Scenario2

A customer, who plans to travel to France, wants a weather report in Paris. As an English speaker, he would like all information translated to English and sent to his email address. Moreover, as he had not booked for a flight yet, he wants to book for the flight departed from London to Paris on 10 June 2008.

Scenario3

A salesman is traveling around the city. There are five clients he plans to meet in one day. It is a good idea to be notified about the distance to the next client office in his current location so that he could calculate for a fuel cost based on the distance coverage. At the end of the day, he wants to send information about the deals he made during the day to his head office and he wants to relax by watching a movie in a theater near his current location.

In each scenario, the end user requests for different service composition in a different circumstance (i.e. user context). This demonstrates a possibility of how the end user could benefit from the IA functionalities in practice. Essentially, we present two results for each user scenario. The two results are: 1) Service compositions in SPATEL and 2) Service inputs extracted from the user request. The discussion part deliberates on how well the IA serves the end-user and also the effectiveness of the IA.

5.1 Three user scenarios: The results

In this section, we present the end-user requests for each scenario together with the IA results. In essence, the end-user either submits a request to IA directly or he configures the services to be invoked by IA in Recommended Service Rule (or the combination of both). For the second case, IA will invoke specified service(s) when certain user context (e.g. user locations) arises.

Result and Discussion

For each scenario, we present two results, these are 1) Service compositions in SPATEL and 2) Service inputs extracted from the user request.

It is worth noting that there is compatibility issue between IA and NLP-tool (if the MRS result from NLP were to be used by IA). The IA, developed in this thesis, supports a request in English. On the other hand, the NLP-tool developed by linguistic experts supports a request in Norwegian. As we have discussed in section 3.5.8, for a simple sentence, IA can translate the MRS result in Norwegian to English by utilizing a dictionary file. However, this implies that we require Norwegian sentences (for each user request) for NLP-tool to process. The equivalences between English sentence and Norwegian sentence are shown in Table 3.

English	Norwegian
Find vegetarian restaurant in my area and send it by sms or email	Finn vegetarrestaurant i mitt område og send den pr. email eller sms
Find weather report in Paris, translated to English and send it by email	Finn værfold i Paris oversatt til engelsk og send pr. sms
Search a flight from London to Paris on 10-06-2008 and book it	søk en flight fra London til Paris for 10-06-2008 og bestill den
Find an address of theater in my area and send it to my email	Finn adressen til et teater I mitt område og send den til min email

Table 3: All user requests in English and Norwegian

5.1.1 Scenario 1

In scenario 1, the end user would like to search for a restaurant. Moreover, he would like particular services (i.e. traffic report, weather report) to be delivered to him depending on his current location. Therefore, the end user instructs IA as follows:

- Initially, the end-user submits a request: “Find vegetarian restaurant in my area and send it by sms or email”. IA could derive the current user location if the context-handling feature had been enabled. Otherwise, the default location will be used. We have made an assumption that this default location is initially known by IA. In practice, the default location of end-user could be configured (e.g. via the interface).
- The end-user also configures Recommended Service Rule as shown in Figure 85. We have made an assumption that the end-user lives in Paris. Moreover, street@telin.nl is an address of the road (where the end user will drive on) and restaurant@telin.nl is an address of the restaurant. This implies that the address of restaurant is known in advance. This is opposed to the real situation where the address of the restaurant has not been known (i.e. the address of the restaurant is obtained from the request “Find vegetarian restaurant in my area and send it by sms or email”. It is not known by the time the user configures the Recommended Service Rule). Hence, in practice, the end user shall configure a list of possible addresses of restaurants in which he may have a dinner (a change in user location to one of the addresses in the list will result in the service composition). Specifically, the end user is driving if current user location is street@telin.nl. Similarly, the weather report will be delivered to the end-user while he is in the restaurant.

This scenario demonstrates IA properties presented in section 3.5.1-3.5.14

! context rule, invoke operation of service when the context occurs			
!			
! Context	Service	operation	input parameter
!			

"street@telin.nl"	"CITYTrafficReport"	"ReportTrafficCondition"	"Paris"
"street@telin.nl"	"MessagingServer"	"SmsSender"	"TrafficReport,00475612605"
"restaurant@telin.nl"	"GlobalWeatherForecast"	"WeatherReport"	"Paris"
"restaurant@telin.nl"	"MessagingServer"	"SmsSender"	"WeatherReport,00475612605"

Figure 85: Recommended Service Rule for scenario 1

5.1.1.1 Result of scenario 1: Service compositions in SPATEL

When the end-user submits a request: “Find vegetarian restaurant in my area and send it by sms or email”, IA construct service compositions, accordingly. There are 100 service compositions in SPATEL generated by IA. The completed result can be found in *spatel* folder in a file attached with this thesis (Appendix F). Also, if the context-handling feature is turned on, IA will construct service compositions according to user locations. The completed result can be found in *spatel2* folder in a file attached with this thesis (Appendix F). It is worth noting that the SPATEL files generated by IA with MRS and the SPATEL files generated by IA without MRS are the same. The different is on the order of the SPATEL files (service compositions) that are generated. For example, the first SPATEL file generated by IA with MRS is the 10th file generated by IA without MRS (section 5.4).

In this section, we explain on samples of a service composition (in SPATEL) generated by IA. This explanation is applied in general for other SPATEL files (i.e. for other service compositions). A SPATEL of the request (“Find vegetarian restaurant in my area and send it by sms or email”) is shown in Figure 86. The SPATEL file represents a service composition of two services that support two operations, namely, SearchVegetarianRestaurant and SendSms. The SPATEL format agrees with what we have presented in section 2.3.2. As mentioned in section 2.3.2, the service composition is represented in a state machine notion. There are four external variables to be used in the state machine. These are SEA, IN1, SEN and OUT1. SEA is used to store a proxy of an operation, SearchVegetarianRestaurant of the first service component. SEN is used to store a proxy of an operation, SendSms of the second service component. IN1 is used to store a value of external input given to the state machine (e.g. “my area”). OUT1 is used to store a value of output produced by the second service operation.

When the input value is obtained (<acceptAction opaqueBody="GetValues(IN1)"/>) from the user (i.e. outside state machine), the state machine transforms from an initial state to the first active state (this is represented in the transition tag). The first active state is to invoke the operation, SearchVegetarianRestaurant, with the input value that has been stored in IN1 (SEA.SearchVegetarianRestaurant(IN1)). The result of the operation, SearchVegetarianRestaurant, is stored in an internal variable, VAR1.

Then, the state machine transforms from the first active state to the second active state (this is represented in the transition tag). The second active state is to invoke the operation, SendSms, with an input obtained from a result of SearchVegetarianRestaurant via VAR1 and another input obtained from the user via IN2 (SEN.SendSms(VAR1,IN2)). The result of SendSms is kept

Result and Discussion

in the variable OUT1. Finally, the state machine transforms to the final state with a result of service composition kept in OUT1.

Essentially, we have verified that the SPATEL is in a format as described in section 2.3.2. Furthermore, it represents the right service composition for the request. For example, SearchVegetarianRestaurant is invoked before SendSms with the right parameters (e.g. the result of SearchVegetarianRestaurant is used as an input of SendSms while SearchVegetarianRestaurant obtains its input from the user etc).

Result and Discussion

```
- <xmi:XMI xmi:version="2.0">
- <spatel:ServiceLibrary name="CompositeService" xmi:id="rgKwKvJmJlMm295WLqPqnBCvJ">
- <nestedPackage name="CompositeService" xmi:id="qA2SOqPqWw+wLCCcCdZ_1zPLhGg" xsi:type="spatel:ServicePackage">
- <service name="CompositeService" xmi:id="ODeJmMm8nB3TWUzXmKKhORN0" xsi:type="spatel:ServiceInterface">
- <ownedOperation name="orchestrate" xmi:id="Pph5151TPStp958tHEHf4+wP" xsi:type="spatel:ServiceOperation">
- <behavior xmi:id="apouqDBDca+azf478Y7XSZ+vKu" xsi:type="spatel:StateMachine">
- <initSection>
  <action opaqueBody="var SEA : SearchVegetarianRestaurant = createProxy("SearchVegetarianRestaurant)" xsi:type="spatel:VariableDeclarationAction"/>
  <action opaqueBody="var IN1 : String" xsi:type="spatel:VariableDeclarationAction"/>
  <action opaqueBody="var SEN : SendSms = createProxy("SendSms")" xsi:type="spatel:VariableDeclarationAction"/>
  <action opaqueBody="var OUT1 : String" xsi:type="spatel:VariableDeclarationAction"/>
</initSection>
- <region>
  <subvertex kind="initial" name="Initial" outgoing="Qfd5f40PXUuBdZy0yMjLMoqB6" xmi:id="xYPrmqFCdCcUuae23Q_QMfeh73" xsi:type="spatel:InitialNode"/>
  <transition name="tr1" source="xYPrmqFCdCcUuae23Q_QMfeh73" target="yFvIF9qB7XTWSsprLEadY+5n7" xmi:id="Qfd5f40PXUuBdZy0yMjLMoqB6"/>
  <subvertex incoming="Qfd5f40PXUuBdZy0yMjLMoqB6" name="Wait" outgoing="WOosoCAC8ZORspwnC9acdz0+O" xmi:id="yFvIF9qB7XTWSsprLEadY+5n7"
  xsi:type="spatel:WaitState"/>
- <transition name="tr2" source="yFvIF9qB7XTWSsprLEadY+5n7" target="DfTrEGHj0QMTtoEpD9Zge1c4y" xmi:id="WOosoCAC8ZORspwnC9acdz0+O">
- <trigger>
  <acceptAction opaqueBody="GetValues(IN1)"/>
  </trigger>
</transition>
  <subvertex incoming="WOosoCAC8ZORspwnC9acdz0+O" kind="accept" name="GetValues(IN1)" outgoing="6_ORkNmrvFBD9dgh5kmhADDeVB"
  xmi:id="DfTrEGHj0QMTtoEpD9Zge1c4y" xsi:type="spatel:AcceptNode"/>
  <transition name="tr3" source="DfTrEGHj0QMTtoEpD9Zge1c4y" target="WmBqEx_PQTOoltGCZCbZg2_2j" xmi:id="6_ORkNmrvFBD9dgh5kmhADDeVB"/>
- <subvertex incoming="6_ORkNmrvFBD9dgh5kmhADDeVB" name="VAR1 = SEA.SearchVegetarianRestaurant(IN1)" outgoing="20PWSuprsHIEJlmi29rGrEFbde"
  xmi:id="WmBqEx_PQTOoltGCZCbZg2_2j" xsi:type="spatel:SyncCallState">
  <callAction opaqueBody="VAR1 = SEA.SearchVegetarianRestaurant(IN1)" xsi:type="spatel:AssignmentAction"/>
</subvertex>
  <transition name="tr4" source="WmBqEx_PQTOoltGCZCbZg2_2j" target="dloCAZ7dac0zL74VNRmDmABF" xmi:id="20PWSuprsHIEJlmi29rGrEFbde"/>
- <subvertex incoming="20PWSuprsHIEJlmi29rGrEFbde" name="OUT1 = SEN.SendSms(VAR1,IN2)" outgoing="TVxsGf_xLPLqikBD5VRtSxYy_"
  xmi:id="dloCAZ7dac0zL74VNRmDmABF" xsi:type="spatel:SyncCallState">
  <callAction opaqueBody="OUT1 = SEN.SendSms(VAR1,IN2)" xsi:type="spatel:AssignmentAction"/>
</subvertex>
  <transition name="tr5" source="dloCAZ7dac0zL74VNRmDmABF" target="FcZ+Yyy+PLnKpmpCuKtHFMme7g" xmi:id="TVxsGf_xLPLqikBD5VRtSxYy_"/>
  <subvertex incoming="TVxsGf_xLPLqikBD5VRtSxYy_" name="Final1" xmi:id="FcZ+Yyy+PLnKpmpCuKtHFMme7g" xsi:type="spatel:FinalState"/>
</region>
  <variable name="VAR1"/>
  <variable name="SEA"/>
  <variable name="SEN"/>
</behavior>
</ownedOperation>
</service>
- <event name="GetValues" xmi:id="T6WSb_w+wKILlMrQm7EBcYbTx" xsi:type="spatel:Service.Event">
  <ownedAttribute instanceType="String" name="IN1" xmi:id="325VXZVz0yMKNJb03_R589ard" xsi:type="spatel:Service.Attribute"/>
</event>
</nestedPackage>
</spatel:ServiceLibrary>
</xmi:XMI>
```

Figure 86: A SPATEL file generated from a request “Find vegetarian restaurant in my area and send it by sms or email”

Result and Discussion

```
- <xmi:XMI xmi:version="2.0">
- <spatel:ServiceLibrary name="CompositeService" xmi:id="G+G8dVv0+OzIFjnVn63U3SPXz">
- <nestedPackage name="CompositeService" xmi:id="+4UBEBcYa_xdMxLEfcyOyO473VR" xsi:type="spatel:ServicePackage">
- <service name="CompositeService" xmi:id="SNok4j75TWUtpwd130QNQMqIE" xsi:type="spatel:ServiceInterface">
- <ownedOperation name="orchestrate" xmi:id="n5zOLpVrtwsHCFhd2d1+OQOnT" xsi:type="spatel:ServiceOperation">
- <behavior xmi:id="p73V9Ccaza+vKNJgg414VCCeB" xsi:type="spatel:StateMachine">
- <initSection>
  <action opaqueBody="var REP : ReportTrafficCondition = createProxy("ReportTrafficCondition")" xsi:type="spatel:VariableDeclarationAction"/>
  <action opaqueBody="var IN1 : String" xsi:type="spatel:VariableDeclarationAction"/>
  <action opaqueBody="var SMS : SmsSender = createProxy("SmsSender")" xsi:type="spatel:VariableDeclarationAction"/>
  <action opaqueBody="var OUT1 : String" xsi:type="spatel:VariableDeclarationAction"/>
</initSection>
- <region>
  <subvertex kind="initial" name="Initial1" outgoing="p3oCFCcYbc1uzPNQJf58oE9BD" xmi:id="_37XCeDbZy_1R0PMmqAp8qJLM" xsi:type="spatel:InitialNode"/>
  <transition name="tr1" source="_37XCeDbZy_1R0PMmqAp8qJLM" target="WmqmWusuFfEgc140MJjJQmjr" xmi:id="p3oCFCcYbc1uzPNQJf58oE9BD"/>
  <subvertex incoming="p3oCFCcYbc1uzPNQJf58oE9BD" name="Wait" outgoing="7SOoksnBDEg+NNQpBqE9GcY+d" xmi:id="WmqmWusuFfEgc140MJjJQmjr"
  xsi:type="spatel:WaitState"/>
- <transition name="tr2" source="WmqmWusuFfEgc140MJjJQmjr" target="zHDfbc1+OuHDfhe2+1+PKOoro" xmi:id="7SOoksnBDEg+NNQpBqE9GcY+d">
- <trigger>
  <acceptAction opaqueBody="GetValues(IN1)"/>
</trigger>
</transition>
  <subvertex incoming="7SOoksnBDEg+NNQpBqE9GcY+d" kind="accept" name="GetValues(IN1)" outgoing="GVTtStpCGHhZgMnpB796WdVzw"
  xmi:id="zHDfbc1+OuHDfhe2+1+PKOoro" xsi:type="spatel:AcceptNode"/>
  <transition name="tr3" source="zHDfbc1+OuHDfhe2+1+PKOoro" target="JAbXZUz1zJKL1TTPtFoHDGeb0" xmi:id="GVTtStpCGHhZgMnpB796WdVzw"/>
- <subvertex incoming="GVTtStpCGHhZgMnpB796WdVzw" name="VAR1 = REP.ReportTrafficCondition(IN1)" outgoing="v+wI24ZRrUwsuMEeEffg62RqFA"
  xmi:id="JAbXZUz1zJKL1TTPtFoHDGeb0" xsi:type="spatel:SyncCallState">
  <callAction opaqueBody="VAR1 = REP.ReportTrafficCondition(IN1)" xsi:type="spatel:AssignmentAction"/>
</subvertex>
  <transition name="tr4" source="JAbXZUz1zJKL1TTPtFoHDGeb0" target="DmjCnB3TAHhZy+_1NJooOk8B7" xmi:id="v+wI24ZRrUwsuMEeEffg62RqFA"/>
- <subvertex incoming="v+wI24ZRrUwsuMEeEffg62RqFA" name="OUT1 = SMS.SmsSender(VAR1,IN2)" outgoing="R2SNRNsknFDEAcYZ+x2k457ZVr"
  xmi:id="DmjCnB3TAHhZy+_1NJooOk8B7" xsi:type="spatel:SyncCallState">
  <callAction opaqueBody="OUT1 = SMS.SmsSender(VAR1,IN2)" xsi:type="spatel:AssignmentAction"/>
</subvertex>
  <transition name="tr5" source="DmjCnB3TAHhZy+_1NJooOk8B7" target="P3TRUQsmPFBaDfac2+NjB69aVv" xmi:id="R2SNRNsknFDEAcYZ+x2k457ZVr"/>
  <subvertex incoming="R2SNRNsknFDEAcYZ+x2k457ZVr" name="Final1" xmi:id="P3TRUQsmPFBaDfac2+NjB69aVv" xsi:type="spatel:FinalState"/>
</region>
  <variable name="VAR1"/>
  <variable name="REP"/>
  <variable name="SMS"/>
</behavior>
</ownedOperation>
</service>
- <event name="GetValues" xmi:id="GPVsoCAC9UAHjf_x+xQTLkh6h7" xsi:type="spatel:ServiceEvent">
  <ownedAttribute instanceType="String" name="IN1" xmi:id="nd142RNRNploFcDbZz0zMJMJP" xsi:type="spatel:ServiceAttribute"/>
</event>
</nestedPackage>
</spatel:ServiceLibrary>
</xmi:XMI>
```

Figure 87: A SPATEL file generated when a user location is changed to street@telin.nl

A SPATEL constructed as a result of a change in user location to street@telin.nl is shown in Figure 87. The SPATEL file represents a service composition of two services that support two operations, namely, ReportTrafficCondition and SmsSender. The SPATEL format agrees with what we have presented in section 2.3.2. As mentioned in section 2.3.2, the service composition is represented in a state machine notion. There are four external variables to be used in the state machine. These are REP, IN1, SMS and OUT1. REP is used to store a proxy of an operation, ReportTrafficCondition of the first service component. SMS is used to store a proxy of an operation, SmsSender of the second service component. IN1 is used to store a value of external input given to the state machine. OUT1 is used to store a value of output produced by the second service operation.

When the input value is obtained (`<acceptAction opaqueBody="GetValues(IN1)"/>`) from the user (i.e. outside state machine), the state machine transforms from an initial state to the first active state (this is represented in the transition tag). The first active state is to invoke the operation, `ReportTrafficCondition`, with the input value that has been stored in `IN1` (`REP.ReportTrafficCondition(IN1)`). The result of the operation, `ReportTrafficCondition`, is stored in an internal variable, `VAR1`.

Then, the state machine transforms from the first active state to the second active state (this is represented in the transition tag). The second active state is to invoke the operation, `SmsSender`, with an input obtained from a result of `ReportTrafficCondition` via `VAR1` and another input obtained from the user via `IN2` (`SMS.SmsSender(VAR1,IN2)`). The result of `SmsSender` is kept in the variable `OUT1`. Finally, the state machine transforms to the final state with a result of service composition kept in `OUT1`.

Essentially, we have verified that the SPATEL is in a format as described in section 2.3.2. Furthermore, it represents the right service composition for the request. For example, `ReportTrafficCondition` is invoked before `SmsSender` with the right parameters (e.g. the result of `ReportTrafficCondition` is used as an input of `SmsSender` while `ReportTrafficCondition` obtains its input from the user etc).

5.1.1.2 Result of scenario 1: Service input parameters

In this section, we present service input parameters extracted from the user request. For the service composition deduced from recommended service rule, the input parameters are obtained directly from the file (`ContextRule.txt`). Therefore, we are not interested in the input parameters for that case. The service input parameters extracted by IA (without MRS) from a user request: “Find vegetarian restaurant in my area and send it by sms or email” is shown in Figure 88.

The syntax is `<Service name, Service goal>, [service input, matching weight]`. For example, we can read the first entry as: an input for a service, `VEGRestaurantFinding`, that has a service goal, `SearchVegetarianRestaurant`, is A2.05@telin.nl. It has 80 percent (i.e. a matching weight is 8 out of 10) chance that A2.05@telin.nl is an input of the service `VEGRestaurantFinding`. Notice that A2.05@telin.nl is derived from a keyword “my area” in the user request. We assume that A2.05@telin.nl is a default current location of the end user. If the context handling feature is turned on, the real user location will be derived (instead of A2.05@telin.nl).

Indeed, some of the input parameters are not accurate. For example, A2.05@telin.nl should not be used as an input of `UKLocation` or `UKLocationPro` that receives a `ZipCode` as an input. This result can be improved by using MRS. As discussed in section 3.5.8, IA verifies the service input parameters matching with MRS of the sentence (i.e. user request). Basically, IA ensures that the services and their inputs in Figure 88 agree with a meaning of the sentence (i.e. does the user really mean to use the word such as “my area” to be an input of a service `UKLocation`).

As shown in Figure 89, service input parameters that fail to agree with MRS have been removed. This makes the input parameters extraction becomes more accurate. However, it is worth noting that, for both cases, IA could not extract service input for the service, `SmsSender` and the service, `MailDaemon`, for example. This is because the user request is not complete. IA could not infer an owner of the mobile phone or email from the sentence (i.e. the sentence does

Result and Discussion

not explicitly state who is the owner of the email). If the request were to be “Find vegetarian restaurant in my area and send it to **my** email”, IA would have been able to extract the service input of MailDaemon. Indeed, this illustrates a limitation of IA for an ambiguous sentence. IA would have guessed from the context of the sentence that the owner of email is the user, himself. This could be a challenge in future work.

```
Inputs from user:
[<VEGRestaurantFind-
ing,SearchVegetarianRestaurant>,<A2.05@telin.nl,8>]][<GetRestaurantByAddress,FindResta
urantByAd-
dress>,<A2.05@telin.nl,8>]][<GetRestaurantByCoords,FindRestaurantByLocation>,<A2.05
@telin.nl,8>]][<UKLocation,GetLocation>,<A2.05@telin.nl,8>]][<UKLocationPro,GetLocati
on>,<A2.05@telin.nl,9>]]
```

Figure 88: Service input parameters extracted from a request “Find vegetarian restaurant in my area and send it by sms or email” (IA without MRS)

```
Inputs from user:
[<VEGRestaurantFind-
ing,SearchVegetarianRestaurant>,<A2.05@telin.nl,8>]][<GetRestaurantByAddress,FindResta
urantByAd-
dress>,<A2.05@telin.nl,8>]][<GetRestaurantByCoords,FindRestaurantByLocation>,<A2.05
@telin.nl,8>]]
```

Figure 89: Service input parameters extracted from a request “Find vegetarian restaurant in my area and send it by sms or email” (IA with MRS)

5.1.2 Scenario 2

In scenario 2, the end user would like to get a weather report in Paris. Moreover, he would like the weather report to be translated to English and send to his email. He also needs to search for a flight and book the flight. Therefore, the end user instructs IA as follows:

- Firstly, the end user submits a request: “Find weather report in Paris, translated to English and send it by email” in order to get the weather report.
- Secondly, the end user submits a request: “Search a flight from London to Paris on 10-06-2008 and book it” in order to book a flight from London (i.e. his current location) to Paris on 10 June 2008

This scenario demonstrates IA properties presented in section 3.5.1-3.5.8.

5.1.2.1 Result of scenario 2: Service compositions in SPATEL

When the end-user submits a request: “Find weather report in Paris, translated to English and send it by email”, IA construct service compositions, accordingly. There are 77 service compositions in SPATEL generated by IA. The completed result can be found in *spatel* folder in a file attached with this thesis (Appendix F). It is worth noting that the SPATEL files generated by IA with MRS and the SPATEL files generated by IA without MRS are the same. The different is on the order of the SPATEL files (service compositions) that are generated.

In this section, we explain on samples of a service composition (in SPATEL) generated by IA. This explanation is applied in general for other SPATEL files (i.e. for other service compositions). A SPATEL of the request (“Find weather report in Paris, translated to English and send it by email”) is shown in Figure 90. The SPATEL file represents a service composition of three services that support three operations, namely, WeatherReport, TranslateToEnglish and SendEmail. The SPATEL format agrees with what we have presented in section 2.3.2. As mentioned in section 2.3.2, the service composition is represented in a state machine notion. There are five external variables to be used in the state machine. These are WEA, IN1, TRA, SEN and OUT1. WEA is used to store a proxy of an operation, WeatherReport of the first service component. TRA is used to store a proxy of an operation, TranslateToEnglish of the second service component. SEN is used to store a proxy of an operation, SendEmail of the third service component. IN1 is used to store a value of external input given to the state machine (e.g. “Paris”). OUT1 is used to store a value of output produced by the third service operation.

When the input value is obtained (`<acceptAction opaqueBody="GetValues(IN1)"/>`) from the user (i.e. outside state machine), the state machine transforms from an initial state to the first active state (this is represented in the transition tag). The first active state is to invoke the operation, WeatherReport, with the input value that has been stored in IN1 (WEA.WeatherReport(IN1)). The result of the operation, WeatherReport, is stored in an internal variable, VAR1.

Then, the state machine transforms from the first active state to the second active state (this is represented in the transition tag). The second active state is to invoke the operation, TranslateToEnglish, with an input obtained from a result of WeatherReport via VAR1 (TRA.TranslateToEnglish(VAR1)). The result of the operation, TranslateToEnglish, is stored in an internal variable, VAR2.

Next, the state machine transforms from the second active state to the third active state (this is represented in the transition tag). The third active state is to invoke the operation, SendEmail, with an input obtained from a result of TranslateToEnglish via VAR2 and another input obtained from the user via IN2 (SEN.SendEmail(VAR2,IN2)). The result of SendEmail is kept in the variable OUT1. Finally, the state machine transforms to the final state with a result of service composition kept in OUT1.

Essentially, we have verified that the SPATEL is in a format as described in section 2.3.2. Furthermore, it represents the right service composition for the request. For example, WeatherReport is invoked before TranslateToEnglish with the right input parameters. Specifically, the output of WeatherReport (weather information) is used as an input of TranslateToEnglish. Similarly, SendEmail receives one input (e.g. the translated text) from TranslateToEnglish and another input (email address) from the user.

Result and Discussion

```
- <xmi:XMI xmi:version="2.0">
- <spatel:ServiceLibrary name="CompositeService" xmi:id="SoNKgFgd151QOVRqu58rFBGCcU">
- <nestedPackage name="CompositeService" xmi:id="le3E3_RTQk764TvtZcY05_Q" xsi:type="spatel:ServicePackage">
- <service name="CompositeService" xmi:id="zrGDbAaXe3_O_N5XaWxsvtHOGk" xsi:type="spatel:ServiceInterface">
- <ownedOperation name="orchestrate" xmi:id="PFhdg51SnB3TbYytpDBbfa+yv" xsi:type="spatel:ServiceOperation">
- <behavior xmi:id="FUX_tHxKS8dCYUy_0QLHmokDn" xsi:type="spatel:StateMachine">
- <mitSection>
  <action opaqueBody="var WEA : WeatherReport = createProxy("WeatherReport")" xsi:type="spatel:VariableDeclarationAction"/>
  <action opaqueBody="var IN1 : String" xsi:type="spatel:VariableDeclarationAction"/>
  <action opaqueBody="var TRA : TranslateToEnglish = createProxy("TranslateToEnglish")" xsi:type="spatel:VariableDeclarationAction"/>
  <action opaqueBody="var SEN : SendEmail = createProxy("SendEmail")" xsi:type="spatel:VariableDeclarationAction"/>
  <action opaqueBody="var OUT1 : String" xsi:type="spatel:VariableDeclarationAction"/>
</mitSection>
- <region>
  <subvertex kind="initial" name="Initial1" outgoing="ezY+uLLmLRyuuJsFCbXa4_1_" xmi:id="t95VCFeb+Z_yIEHi4f2_7XAE" xsi:type="spatel:InitialNode"/>
  <transition name="tr1" source="t95VCFeb+Z_yIEHi4f2_7XAE" target="MZVwtwRLMNm8WXZ+_2yNUPuUuq" xmi:id="ezY+uLLmLRyuuJsFCbXa4_1_"/>
  <subvertex incoming="ezY+uLLmLRyuuJsFCbXa4_1_" name="War" outgoing="8j79Bfae183A6bcd0b_noIFGH" xmi:id="MZVwtwRLMNm8WXZ+_2yNUPuUuq"
  xsi:type="spatel:WaitState"/>
- <transition name="tr2" source="MZVwtwRLMNm8WXZ+_2yNUPuUuq" target="wWwvzRMTLhqE9GCcef3SrytHw" xmi:id="8j79Bfae183A6bcd0b_noIFGH">
- <trigger>
  <acceptAction opaqueBody="GetValues(IN1)"/>
</trigger>
</transition>
<subvertex incoming="8j79Bfae183A6bcd0b_noIFGH" kind="accept" name="GetValues(IN1)" outgoing="Jea0y_yM+5VQXUyzwLIMGlog44"
xmi:id="wWwvzRMTLhqE9GCcef3SrytHw" xsi:type="spatel:AcceptNode"/>
<transition name="tr3" source="wWwvzRMTLhqE9GCcef3SrytHw" target="_EHie1e1d184W5VNsusGBDEdki" xmi:id="Jea0y_yM+5VQXUyzwLIMGlog44"/>
- <subvertex incoming="Jea0y_yM+5VQXUyzwLIMGlog44" name="VAR1 = WEA.WeatherReport(IN1)" outgoing="+h4f958YTaWyt+OGgHIPXstwtl"
xmi:id="_EHie1e1d184W5VNsusGBDEdki" xsi:type="spatel:SyncCallState">
  <callAction opaqueBody="VAR1 = WEA.WeatherReport(IN1)" xsi:type="spatel:AssignmentAction"/>
</subvertex>
<transition name="tr4" source="_EHie1e1d184W5VNsusGBDEdki" target="+osHDdCca+0+NJMInh25pEAD9a" xmi:id="+h4f958YTaWyt+OGgHIPXstwtl"/>
- <subvertex incoming="+h4f958YTaWyt+OGgHIPXstwtl" name="VAR2 = TRA.TranslateToEnglish(VAR1)" outgoing="UqnpDEICdZc0nqB68YbdZ+wxN"
xmi:id="+osHDdCca+0+NJMInh25pEAD9a" xsi:type="spatel:SyncCallState">
  <callAction opaqueBody="VAR2 = TRA.TranslateToEnglish(VAR1)" xsi:type="spatel:AssignmentAction"/>
</subvertex>
<transition name="tr5" source="+osHDdCca+0+NJMInh25pEAD9a" target="rBEAaWbXxe41R_QRUqmErFAHc" xmi:id="UqnpDEICdZc0nqB68YbdZ+wxN"/>
- <subvertex incoming="UqnpDEICdZc0nqB68YbdZ+wxN" name="OUT1 = SEN.SendEmail(VAR2,IN2)" outgoing="Xxd1c415TQqPpruHHdGgOWwswr"
xmi:id="rBEAaWbXxe41R_QRUqmErFAHc" xsi:type="spatel:SyncCallState">
  <callAction opaqueBody="OUT1 = SEN.SendEmail(VAR2,IN2)" xsi:type="spatel:AssignmentAction"/>
</subvertex>
<transition name="tr6" source="rBEAaWbXxe41R_QRUqmErFAHc" target="UMTPpx_LPKkPpUqysLzKLMmmpI" xmi:id="Xxd1c415TQqPpruHHdGgOWwswr"/>
<subvertex incoming="Xxd1c415TQqPpruHHdGgOWwswr" name="Final1" xmi:id="UMTPpx_LPKkPpUqysLzKLMmmpI" xsi:type="spatel:FinalState"/>
</region>
<variable name="VAR1"/>
<variable name="VAR2"/>
<variable name="WEA"/>
<variable name="TRA"/>
<variable name="SEN"/>
</behavior>
</ownedOperation>
</service>
- <event name="GetValues" xmi:id="_aWZXxvzLDdGd15n84U4VStru" xsi:type="spatel:ServiceEvent">
  <ownedAttribute instanceType="String" name="IN1" xmi:id="BQUuY+XyvOPQTPpkHCJBbiQuS" xsi:type="spatel:ServiceAttribute"/>
</event>
</nestedPackage>
</spatel:ServiceLibrary>
</xmi:XMI>
```

Result and Discussion

Figure 90: A SPATEL file generated from a request “Find weather report in Paris, translated to English and send it by email”

```
- <xmi:XMI xmi:version="2.0">
- <spatel:ServiceLibrary name="CompositeService" xmi:id="dW7vn_EN92FTPZLbh_gYm4xAs">
- <nestedPackage name="CompositeService" xmi:id="9OGUGVMZpaRgq7+lv9OF1AULa_" xsi:type="spatel:ServicePackage">
- <service name="CompositeService" xmi:id="A0GUQaPdVfzebp2B8p7HWMY6LB" xsi:type="spatel:ServiceInterface">
- <ownedOperation name="orchestrate" xmi:id="5KYIAPetZVjy62H+C4Iu3xAP90" xsi:type="spatel:ServiceOperation">
- <behavior xmi:id="zp1HWM7MbSgQfWqzlcq4kqg3H" xsi:type="spatel:StateMachine">
- <initSection>
  <action opaqueBody="var SEA : SearchFlight = createProxy("SearchFlight")" xsi:type="spatel:VariableDeclarationAction"/>
  <action opaqueBody="var IN1 : String" xsi:type="spatel:VariableDeclarationAction"/>
  <action opaqueBody="var IN2 : String" xsi:type="spatel:VariableDeclarationAction"/>
  <action opaqueBody="var IN3 : String" xsi:type="spatel:VariableDeclarationAction"/>
  <action opaqueBody="var BOO : BookFlight = createProxy("BookFlight")" xsi:type="spatel:VariableDeclarationAction"/>
  <action opaqueBody="var OUT1 : String" xsi:type="spatel:VariableDeclarationAction"/>
</initSection>
- <region>
  <subvertex kind="initial" name="Initial1" outgoing="J5KYQfPdVlyiao0G8r7IXOCQIT" xmi:id="CzCRqcShr9_mwE4K_BUEXTWkfq" xsi:type="spatel:InitialNode"/>
  <transition name="tr1" source="CzCRqcShr9_mwE4K_BUEXTWkfq" target="x51Emw91F_E6KYKBPfjZjyp6s" xmi:id="J5KYQfPdVlyiao0G8r7IXOCQIT"/>
  <subvertex incoming="J5KYQfPdVlyiao0G8r7IXOCQIT" name="Wait" outgoing="0FxA0GVH7LaXNgunwYhxo0n0u" xmi:id="x51Emw91F_E6KYKBPfjZjyp6s" xsi:type="spatel:WaitState"/>
- <transition name="tr2" source="x51Emw91F_E6KYKBPfjZjyp6s" target="bqhv8vm_9O80GThZLWpfvKeTku" xmi:id="0FxA0GVH7LaXNgunwYhxo0n0u">
- <trigger>
  <acceptAction opaqueBody="GetValues(IN1,IN2,IN3)"/>
</trigger>
</transition>
  <subvertex incoming="0FxA0GVH7LaXNgunwYhxo0n0u" kind="accept" name="GetValues(IN1,IN2,IN3)" outgoing="fpgv8vm+DTJ4KTQd9JESiOK Y1_" xmi:id="bqhv8vm_9O80GThZLWpfvKeTku" xsi:type="spatel:AcceptNode"/>
  <transition name="tr3" source="bqhv8vm_9O80GThZLWpfvKeTku" target="TXIVRbq3wg+7_D3w9IY5zBPdZJ" xmi:id="fpgv8vm+DTJ4KTQd9JESiOK Y1_">
- <subvertex incoming="fpgv8vm+DTJ4KTQd9JESiOK Y1_" name="VAR1,VAR2,VAR3 = SEA.SearchFlight(IN1,IN2,IN3)" outgoing="UjaoeUxzBsp1F5P51BVYObTeu" xmi:id="TXIVRbq3wg+7_D3w9IY5zBPdZJ" xsi:type="spatel:SyncCallState">
  <callAction opaqueBody="VAR1,VAR2,VAR3 = SEA.SearchFlight(IN1,IN2,IN3)" xsi:type="spatel:AssignmentAction"/>
</subvertex>
  <transition name="tr4" source="TXIVRbq3wg+7_D3w9IY5zBPdZJ" target="T8ScjrThwo0PMZn1ftfo0AyA5J" xmi:id="UjaoeUxzBsp1F5P51BVYObTeu"/>
- <subvertex incoming="UjaoeUxzBsp1F5P51BVYObTeu" name="OUT1 = BOO.BookFlight(VAR1,VAR2,IN4)" outgoing="VFAOfogSguq2njxAKDyG6KYJBR" xmi:id="T8ScjrThwo0PMZn1ftfo0AyA5J" xsi:type="spatel:SyncCallState">
  <callAction opaqueBody="OUT1 = BOO.BookFlight(VAR1,VAR2,IN4)" xsi:type="spatel:AssignmentAction"/>
</subvertex>
  <transition name="tr5" source="T8ScjrThwo0PMZn1ftfo0AyA5J" target="Ys3rez5LDzBQIWGUMbpZRgw8+m" xmi:id="VFAOfogSguq2njxAKDyG6KYJBR"/>
  <subvertex incoming="VFAOfogSguq2njxAKDyG6KYJBR" name="Final1" xmi:id="Ys3rez5LDzBQIWGUMbpZRgw8+m" xsi:type="spatel:FinalState"/>
</region>
  <variable name="VAR1"/>
  <variable name="VAR2"/>
  <variable name="VAR3"/>
  <variable name="SEA"/>
  <variable name="BOO"/>
</behavior>
</ownedOperation>
</service>
- <event name="GetValues" xmi:id="hNFTnwtDs0t5t5zCQB2GajbLbr" xsi:type="spatel:Service.Event">
  <ownedAttribute instanceType="String" name="IN1" xmi:id="dn0n_sAOGxFTJZJXPetNDTtw0U" xsi:type="spatel:Service.Attribute"/>
  <ownedAttribute instanceType="String" name="IN2" xmi:id="yp3HVN8JcSgBSHVkXhdm0mdt5K" xsi:type="spatel:Service.Attribute"/>
  <ownedAttribute instanceType="String" name="IN3" xmi:id="RfX2BVLdZRgughv8ur3IR08N" xsi:type="spatel:Service.Attribute"/>
</event>
</nestedPackage>
</spatel:ServiceLibrary>
</xmi:XMI>
```

Figure 91: A SPATEL file generated from a request “Search a flight from London to Paris on 10-06-2008 and book it”

When the end-user submits a request: “Search a flight from London to Paris on 10-06-2008 and book it”, IA constructs service compositions, accordingly. There are 8 service compositions in SPATEL generated by IA (these files are stored in *spatel* folder of the software package attached with this thesis). A SPATEL of the request (“Search a flight from London to Paris on 10-06-2008 and book it”) is shown in Figure 91. The SPATEL file represents a service composition of two services that support two operations, namely, SearchFlight and BookFlight. The SPATEL format agrees with what we have presented in section 2.3.2. As mentioned in section 2.3.2, the service composition is represented in a state machine notion. There are six external variables to be used in the state machine. These are SEA, IN1, IN2, IN3, BOO and OUT1. SEA is used to store a proxy of an operation, SearchFlight of the first service component. BOO is used to store a proxy of an operation, BookFlight of the second service component. IN1, IN2 and IN3 are used to store values of external inputs given to the state machine (e.g. “Paris”, “London”). OUT1 is used to store a value of output produced by the second service operation.

When the input value is obtained (<acceptAction opaqueBody="GetValues(IN1)"/>) from the user (i.e. outside state machine), the state machine transforms from an initial state to the first active state (this is represented in the transition tag). The first active state is to invoke the operation, SearchFlight, with the input values that have been stored in IN1, IN2 and IN3 (SEA.SearchFlight(IN1,IN2,IN3)). The results of the operation, SearchFlight, are stored in the internal variables, VAR1, VAR2 and VAR3 (note that SearchFlight produces three outputs as defined in the service repository).

Then, the state machine transforms from the first active state to the second active state (this is represented in the transition tag). The second active state is to invoke the operation, BookFlight, with two inputs obtained from a result of SearchFlight via VAR1 and VAR2 as well as an input obtained from the user via IN4 (BOO.BookFlight(VAR1,VAR2,IN4)). The result of SearchFlight is kept in the variable OUT1. It is worth noting that the value VAR3 is not used anywhere and it has been ignored. Finally, the state machine transforms to the final state with a result of service composition kept in OUT1.

Essentially, we have verified that the SPATEL is in a format as described in section 2.3.2. Furthermore, it represents the right service composition for the request with the right parameters. For example, SearchFlight is invoked before the BookFlight. Moreover, two outputs of SearchFlight (flight name, departure time) are used as inputs of BookFlight. Also, one input (departure date) of BookFlight is retrieved from the user.

5.1.2.2 Result of scenario 2: Service input parameters

In this section, we present service input parameters extracted from the user request. For the service composition deduced from recommended service rule, the input parameters are obtained directly from the file (ContextRule.txt). Therefore, we are not interested in the input parameters for that case. The service input parameters extracted by IA (without MRS) from a user request: “Find weather report in Paris, translated to English and send it by email” is shown in Figure 92.

The syntax is <Service name, Service goal>, [service input, matching weight]. For example, we can read the first entry as: an input for a service, GlobalWeatherForecast, that has a service goal, WeatherReport, is Paris. It has 90 percent (i.e. a matching weight is 9 out of 10) chance that Paris is an input of the service WeatherReport.

Result and Discussion

Indeed, some of the input parameters are not accurate. For example, Paris should not be used as an input of SkyScanner that receives a depart city and arrival city as inputs (although Paris is a city but, from a user request, it is not a destination of any flights). This result can be improved by using MRS. As discussed in section 3.5.8, IA verifies the service input parameters matching with MRS of the sentence (i.e. user request). Basically, IA ensures that the services and their inputs in Figure 92 agree with a meaning of the sentence.

As shown in Figure 93, service input parameters that fail to agree with MRS have been removed. This makes the input parameters extraction becomes more accurate. However, it is worth noting that, for both cases, IA could not extract service input for the service, MailDaemon, for example. This is because the user request is not complete. IA could not infer an owner of the email from the sentence (i.e. the sentence does not explicitly state who is the owner of the email). If the request were to be “Find weather report in Paris, translated to English and send it to **my** email”, IA would have been able to extract the service input of MailDaemon. Indeed, this illustrates a limitation of IA for an ambiguous sentence. IA would have guessed from the context of the sentence that the owner of email is the user, himself. This could be a challenge in future work.

```
Inputs from user:
[<GlobalWeatherFore-
cast,WeatherReport>,<Paris,9>][<WeatherForecast,WeatherForecast>,<Paris,9>][<Weather
ByCity,WeatherForecast>,<Paris,9>][<SkyScanner,SearchFlight>,<Paris,9>,<Paris,9>]
```

Figure 92: Service input parameters extracted from a request “Find weather report in Paris, translated to English and send it by email” (IA without MRS)

```
Inputs from user:
[<GlobalWeatherFore-
cast,WeatherReport>,<Paris,9>][<WeatherForecast,WeatherForecast>,<Paris,9>][<Weather
ByCity,WeatherForecast>,<Paris,9>]
```

Figure 93: Service input parameters extracted from a request “Find weather report in Paris, translated to English and send it by email” (IA with MRS)

The service input parameters extracted by IA (without MRS) from a user request: “Find weather report in Paris, translated to English and send it by email” is shown in Figure 94.

The syntax is <Service name, Service goal>, [service input, matching weight]. For example, we can read the first entry as: an input for a service, FlightBooking, that has a service goal, BookFlight, is 10-06-2008. It has 90 percent (i.e. a matching weight is 9 out of 10) chance that 10-06-2008 is an input of the service FlightBooking.

Indeed, some of the input parameters are not accurate. For example, Paris should not be used as a departure city of SkyScanner (rather, Paris is an arrival city as defined in the user request). This result can be improved by using MRS. As discussed in section 3.5.8, IA verifies the service input parameters matching with MRS of the sentence (i.e. user request). Basically, IA ensures that the services and their inputs in Figure 94 agree with a meaning of the sentence.

```
Inputs from user:
```

Result and Discussion

```
[<FlightBooking,BookFlight>,[<10-06-2008,9>]] [<SkyScanner,SearchFlight>,[<London,9>,<Paris,9>,<London,9>,<Paris,9>,<10-06-2008,9>]]
```

Figure 94: Service input parameters extracted from a request “Search a flight from London to Paris on 10-06-2008 and book it” (IA without MRS)

Inputs from user:

```
[<FlightBooking,BookFlight>,[<10-06-2008,9>]] [<SkyScanner,SearchFlight>,[<London,9>,<Paris,9>,<10-06-2008,9>]]
```

Figure 95: Service input parameters extracted from a request “Search a flight from London to Paris on 10-06-2008 and book it” (IA with MRS)

As shown in Figure 95, service input parameters that fail to agree with MRS have been removed. This makes the input parameters extraction becomes more accurate. Indeed, the input parameters matching shown in Figure 95 is a perfect result. We extract all inputs from the user request and put them into the right place. The departure date (10-06-2008) is correctly put into FlightBooking (for other inputs of FlightBooking, they are obtained from other service components as described in section 5.1.2.1). The departure city (London), arrival city (Paris) and departure date (10-06-2008) are correctly put into SkyScanner. This demonstrates an efficiency of IA with an unambiguous sentence.

5.1.3 Scenario 3

In scenario 3, the end user would like to search for a theater. Moreover, he would like particular services (i.e. location distance, send SMS) to be delivered to him depending on his current location. Therefore, the end user instructs IA as follows:

- Firstly, the end user submits a request (e.g. at the end of the day): “Find an address of theater in my area and send it to my email” in order to get the theater information. We have made an assumption that the email is an email address of the end user. This email address is initially known by IA. In practice, the email address of the end-user should be configurable (e.g. via the interface). Furthermore, IA could derive the current user location if the context-handling feature had been enabled. Otherwise, the default location will be used. We have made an assumption that this default location is initially known by IA. In practice, the default location of the end-user should be configurable (e.g. via the interface)
- The end-user also configures Recommended Service Rule as shown in Figure 96. Therefore, when the end user (i.e. the salesman) is arrived at one customer’s office, the distance between this current location (of the current customer’s office) and the next location (where he will visit the next customer) are calculated. Lastly, suppose that the customer office E is the final place where the end user plans to visit, the end user specifies his intention to send a deal (in document file) he made during the day to the company’s email address. This is shown in Figure 97

```
! context rule, invoke operation of service when the context occurs
!
! Context          Service          operation          input parameter
!
```


Result and Discussion

"customer_siteA@telin.nl"	"DistanceService"	"GetDistance"	"cus-
tommer_siteA@telin.nl,customer_siteB@telin.nl"			
"customer_siteA@telin.nl"	"MessagingServer"	"SmsSender"	"Dis-
tance,00475612605"			
"customer_siteB@telin.nl""DistanceService"		"GetDistance"	"cus-
tommer_siteB@telin.nl,customer_siteC@telin.nl"			
"customer_siteB@telin.nl""MessagingServer"		"SmsSender"	"Dis-
tance,00475612605"			
"customer_siteC@telin.nl""DistanceService"		"GetDistance"	"cus-
tommer_siteC@telin.nl,customer_siteD@telin.nl"			
"customer_siteC@telin.nl""MessagingServer"		"SmsSender"	"Dis-
tance,00475612605"			
"customer_siteD@telin.nl"	"DistanceService"	"GetDistance"	"cus-
tommer_siteD@telin.nl,customer_siteE@telin.nl"			
"customer_siteD@telin.nl"	"MessagingServer"	"SmsSender"	"Dis-
tance,00475612605"			

Figure 96: Recommended Service Rule for scenario 3

! context rule, invoke operation of service when the context occurs			
!			
! Context	Service	operation	input parameter
!			

"customer_siteE@telin.nl"	"MessagingServer"	"SMTPServer"	
"deal.doc,office@telin.nl"			

Figure 97: Recommended Service Rule for scenario 3 (continue)

This scenario demonstrates IA properties presented in section 3.5.1-3.5.14.

5.1.3.1 Result of scenario 3: Service compositions in SPATEL

When the end-user submits a request: “Find an address of theater in my area and send it to my email”, IA construct service compositions, accordingly. There are 128 service compositions in SPATEL generated by IA. The completed result can be found in *spatel* folder in a file attached with this thesis (Appendix F). Also, if the context-handling feature is turned on, IA will construct service compositions according to user locations. The completed result can be found in *spatel2* folder in the software package attached with this thesis (Appendix F). It is worth noting that the SPATEL files generated by IA with MRS and the SPATEL files generated by IA without MRS are the same. The different is on the order of the SPATEL files (service compositions) that are generated.

In this section, we explain on samples of a service composition (in SPATEL) generated by IA. This explanation is applied in general for other SPATEL files (i.e. for other service compositions). A SPATEL of the request (“Find an address of theater in my area and send it to my email”) is shown in Figure 98. The SPATEL file represents a service composition of two service components that support two operations, namely, SearchTheaterByAddress and SendEmail. The SPATEL format agrees with what we have presented in section 2.3.2. As mentioned in section 2.3.2, the service composition is represented in a state machine notion. There are four external variables to be used in the state machine. These are SEA, IN1, SEN and OUT1. SEA is used to store a proxy of an operation, SearchTheaterByAddress of the first service component. SEN is used to store a proxy of an operation, SendEmail of the second service

Result and Discussion

component. IN1 is used to store a value of an external input given to the state machine (e.g. “my area”). OUT1 is used to store a value of an output produced by the second service operation.

When the input value is obtained (`<acceptAction opaqueBody="GetValues(IN1)"/>`) from the user (i.e. outside state machine), the state machine transforms from an initial state to the first active state (this is represented in the transition tag). The first active state is to invoke the operation, `SearchTheaterByAddress`, with the input value that has been stored in IN1 (`SEA.SearchTheaterByAddress(IN1)`). The result of the operation, `SearchTheaterByAddress`, is stored in an internal variable, VAR1.

Then, the state machine transforms from the first active state to the second active state (this is represented in the transition tag). The second active state is to invoke the operation, `SendEmail`, with an input obtained from a result of `SearchTheaterByAddress` via VAR1 and an input obtained from the user via IN2 (`SEN.SendEmail(VAR1,IN2)`). The result of `SendEmail` is kept in the variable OUT1. Finally, the state machine transforms to the final state with a result of service composition kept in OUT1.

Essentially, we have verified that the SPATEL is in a format as described in section 2.3.2. Furthermore, it represents the right service composition for the request. For example, `SearchTheaterByAddress` is invoked before `SendEmail` with the right input parameters. Specifically, the output of `SearchTheaterByAddress` (theater information) is used as an input of `SendEmail`. Moreover, `SendEmail` receives another input (email address) from the user.

Result and Discussion

```
- <xmi:XMI xmi:version="2.0">
- <spatel:ServiceLibrary name="CompositeService" xmi:id="iftuTdsj0ne03FCy8MbTDRhYn">
- <nestedPackage name="CompositeService" xmi:id="TLZoUQes6+fu8J4wAPdUFUktqa" xsi:type="spatel:ServicePackage">
- <service name="CompositeService" xmi:id="ZoYQfp7ohx4OFzETKZJXSdtenk" xsi:type="spatel:ServiceInterface">
- <ownedOperation name="orchestrate" xmi:id="Mccqgwhwn0E+9DXP9PdTnXOcr4" xsi:type="spatel:ServiceOperation">
- <behavior xmi:id="zpZf0u8u62GVB3HbRgRfVk+kau" xsi:type="spatel:State.Machine">
- <initSection>
- <action opaqueBody="var SEA : SearchTheaterByAddress = createProxy("SearchTheaterByAddress")" xsi:type="spatel:VariableDeclarationAction"/>
- <action opaqueBody="var IN1 : String" xsi:type="spatel:VariableDeclarationAction"/>
- <action opaqueBody="var SEN : SendEmail = createProxy("SendEmail")" xsi:type="spatel:VariableDeclarationAction"/>
- <action opaqueBody="var OUT1 : String" xsi:type="spatel:VariableDeclarationAction"/>
- </initSection>
- <region>
- <subvertex kind="initial" name="Initial" outgoing="At6MQCQRHFWWNbv2whx4LA0E6F" xmi:id="PWThScYm+kqg3J9vDNJXSKZkV" xsi:type="spatel:InitialNode"/>
- <transition name="tr1" source="PWThScYm+kqg3J9vDNJXSKZkV" target="WmcqfVlv9wn_J8O4JZPdPeWXri" xmi:id="At6MQCQRHFWWNbv2whx4LA0E6F"/>
- <subvertex incoming="At6MQCQRHFWWNbv2whx4LA0E6F" name="Wait" outgoing="WJ9NcYhXlcr5rhwD3J3GCLc61E" xmi:id="WmcqfVlv9wn_J8O4JZPdPeWXri" xsi:type="spatel:WaitState"/>
- <transition name="tr2" source="WmcqfVlv9wn_J8O4JZPdPeWXri" target="4KA_D4XJB0cmTiwo0s_t6Kqm" xmi:id="WJ9NcYhXlcr5rhwD3J3GCLc61E">
- <trigger>
- <acceptAction opaqueBody="GetValues(IN1)"/>
- </trigger>
- </transition>
- <subvertex incoming="WJ9NcYhXlcr5rhwD3J3GCLc61E" kind="accept" name="GetValues(IN1)" outgoing="kvrWm+w4r4z9TJ2JYheOdnfjZ" xmi:id="4KA_D4XJB0cmTiwo0s_t6Kqm" xsi:type="spatel:AcceptNode"/>
- <transition name="tr3" source="4KA_D4XJB0cmTiwo0s_t6Kqm" target="ZRFzqbl2v7u8HERB3JXmSKYrhw" xmi:id="kvrWm+w4r4z9TJ2JYheOdnfjZ"/>
- <subvertex incoming="kvrWm+w4r4z9TJ2JYheOdnfjZ" name="VAR1 = SEA.SearchTheaterByAddress(IN1)" outgoing="hvn1E_r9IYQv74HWG7LfpGsguq" xmi:id="ZRFzqbl2v7u8HERB3JXmSKYrhw" xsi:type="spatel:SyncCallState">
- <callAction opaqueBody="VAR1 = SEA.SearchTheaterByAddress(IN1)" xsi:type="spatel:AssignmentAction"/>
- </subvertex>
- <transition name="tr4" source="ZRFzqbl2v7u8HERB3JXmSKYrhw" target="4FUGVLapZPjxo0n0u2M6+CQEXC" xmi:id="hvn1E_r9IYQv74HWG7LfpGsguq"/>
- <subvertex incoming="hvn1E_r9IYQv74HWG7LfpGsguq" name="OUT1 = SEN.SendEmail(VAR1,IN2)" outgoing="Pvmb10r9vnmwBQwm3D9N6KHQgQ" xmi:id="4FUGVLapZPjxo0n0u2M6+CQEXC" xsi:type="spatel:SyncCallState">
- <callAction opaqueBody="OUT1 = SEN.SendEmail(VAR1,IN2)" xsi:type="spatel:AssignmentAction"/>
- </subvertex>
- <transition name="tr5" source="4FUGVLapZPjxo0n0u2M6+CQEXC" target="8v9OYRCMgWmRhw0zShs5yn_r5L" xmi:id="Pvmb10r9vnmwBQwm3D9N6KHQgQ"/>
- <subvertex incoming="hvn1E_r9IYQv74HWG7LfpGsguq" name="OUT1 = SEN.SendEmail(VAR1,IN2)" outgoing="Pvmb10r9vnmwBQwm3D9N6KHQgQ" xmi:id="4FUGVLapZPjxo0n0u2M6+CQEXC" xsi:type="spatel:SyncCallState">
- <callAction opaqueBody="OUT1 = SEN.SendEmail(VAR1,IN2)" xsi:type="spatel:AssignmentAction"/>
- </subvertex>
- <transition name="tr5" source="4FUGVLapZPjxo0n0u2M6+CQEXC" target="8v9OYRCMgWmRhw0zShs5yn_r5L" xmi:id="Pvmb10r9vnmwBQwm3D9N6KHQgQ"/>
- <subvertex incoming="Pvmb10r9vnmwBQwm3D9N6KHQgQ" name="Final" xmi:id="8v9OYRCMgWmRhw0zShs5yn_r5L" xsi:type="spatel:FinalState"/>
- </region>
- <variable name="VAR1"/>
- <variable name="SEA"/>
- <variable name="SEN"/>
- </behavior>
- </ownedOperation>
- </service>
- <event name="GetValues" xmi:id="WhxA0m0F7G6KBQghFH7LHVHqgv" xsi:type="spatel:ServiceEvent">
- <ownedAttribute instanceType="String" name="IN1" xmi:id="7IAQe73DSgcIXmiwgYm+w8u7_D" xsi:type="spatel:ServiceAttribute"/>
- </event>
- </nestedPackage>
- </spatel:ServiceLibrary>
- </xmi:XMI>
```

Figure 98: A SPATEL file generated from a request “Find an address of theater in my area and send it to my email”

Result and Discussion

```
- <xmi:XMI xmi:version="2.0">
- <spatel:ServiceLibrary name="CompositeService" xmi:id="MheOdUxjto0E6Zr3Hyu6MVSCR">
- <nestedPackage name="CompositeService" xmi:id="p2apyq2p3w8N8_EPhOLUkypbp2" xsi:type="spatel:ServicePackage">
- <service name="CompositeService" xmi:id="Ev8MJSI9OcrXTth41i0s6K5E1A" xsi:type="spatel:ServiceInterface">
- <ownedOperation name="orchestrate" xmi:id="jVLZt1ufy6+Dw9NFVDMgqRhm" xsi:type="spatel:ServiceOperation">
- <behavior xmi:id="RgueWlzC4q3IAL9KZRFPVjzb" xsi:type="spatel:State.Machine">
- <initSection>
  <action opaqueBody="var SMT : SMTPServer = createProxy("SMTPServer")" xsi:type="spatel:VariableDeclarationAction"/>
  <action opaqueBody="var IN1 : String" xsi:type="spatel:VariableDeclarationAction"/>
  <action opaqueBody="var IN2 : String" xsi:type="spatel:VariableDeclarationAction"/>
  <action opaqueBody="var OUT1 : String" xsi:type="spatel:VariableDeclarationAction"/>
</initSection>
- <region>
  <subvertex kind="initial" name="Initial1" outgoing="BQgWITrxMESsmZn0u1p4D9J4I" xmi:id="dsYUdy91QgplHzq3I2w4NXT9O" xsi:type="spatel:InitialNode"/>
  <transition name="tr1" source="dsYUdy91QgplHzq3I2w4NXT9O" target="wEvA_GUF7LXhBRcXThZo0met6" xmi:id="BQgWITrxMESsmZn0u1p4D9J4I"/>
  <subvertex incoming="BQgWITrxMESsmZn0u1p4D9J4I" name="Wait" outgoing="jbtFzt3KUQt6KaSBRgqmShvn0" xmi:id="wEvA_GUF7LXhBRcXThZo0met6" xsi:type="spatel:WaitState"/>
- <transition name="tr2" source="wEvA_GUF7LXhBRcXThZo0met6" target="s5r4yAPvnwESK4JTPZYm_gdm0" xmi:id="jbtFzt3KUQt6KaSBRgqmShvn0">
- <trigger>
  <acceptAction opaqueBody="GetValues(IN1,IN2)"/>
</trigger>
</transition>
<subvertex incoming="jbtFzt3KUQt6KaSBRgqmShvn0" kind="accept" name="GetValues(IN1,IN2)" outgoing="rt7M1+7RbXDTgxpZoxu1p1z6Tg" xmi:id="s5r4yAPvnwESK4JTPZYm_gdm0" xsi:type="spatel:AcceptNode"/>
<transition name="tr3" source="s5r4yAPvnwESK4JTPZYm_gdm0" target="A2GVF7OYnrkqu2q+B1G1G8MaK" xmi:id="rt7M1+7RbXDTgxpZoxu1p1z6Tg"/>
- <subvertex incoming="rt7M1+7RbXDTgxpZoxu1p1z6Tg" name="OUT1 = SMT.SMTPServer(IN1,IN2)" outgoing="DRNAJ9_FThZJYnetZp0v6v8HAN" xmi:id="A2GVF7OYnrkqu2q+B1G1G8MaK" xsi:type="spatel:SyncCallState">
  <callAction opaqueBody="OUT1 = SMT.SMTPServer(IN1,IN2)" xsi:type="spatel:AssignmentAction"/>
</subvertex>
<transition name="tr4" source="A2GVF7OYnrkqu2q+B1G1G8MaK" target="FUFULZoaQfv7+g+7NF_EPKVGVk" xmi:id="DRNAJ9_FThZJYnetZp0v6v8HAN"/>
<subvertex incoming="DRNAJ9_FThZJYnetZp0v6v8HAN" name="Final1" xmi:id="FUFULZoaQfv7+g+7NF_EPKVGVk" xsi:type="spatel:FinalState"/>
</region>
<variable name="SMT"/>
</behavior>
</ownedOperation>
</service>
- <event name="GetValues" xmi:id="nZizCgcq0w6v7_DSENYnePYsj" xsi:type="spatel:ServiceEvent">
  <ownedAttribute instanceType="String" name="IN1" xmi:id="y7LHzdSTfWmWnrcrcnitA0nyG5M2" xsi:type="spatel:ServiceAttribute"/>
  <ownedAttribute instanceType="String" name="IN2" xmi:id="fSgbl_ids4J_x5K YQCpFVfPLV" xsi:type="spatel:ServiceAttribute"/>
</event>
</nestedPackage>
</spatel:ServiceLibrary>
</xmi:XMI>
```

Figure 99: A SPATEL file generated when a user location is changed to customer_siteE@telin.nl

A SPATEL constructed as a result of a change in user location to customer_siteE@telin.nl is shown in Figure 99. The SPATEL file represents a service composition of one service that supports only one operation, namely, SMTPServer (this operation is used to send a media file to a destination). The SPATEL format agrees with what we have presented in section 2.3.2.

As mentioned in section 2.3.2, the service composition is represented in a state machine notion. There are four external variables to be used in the state machine. These are SMT, IN1, IN2 and OUT1. SMT is used to store a proxy of an operation, SMTPServer of the first service component. IN1 and IN2 are used to store values of external inputs given to the state machine. OUT1 is used to store a value of output produced by the service operation.

When the input value is obtained (`<acceptAction opaqueBody=" GetValues(IN1,IN2)"/>`) from the user (i.e. outside state machine), the state machine transforms from an initial state to the first active state (this is represented in the transition tag). The first active state is to invoke the operation, `SMTPTServer`, with the input values that have been stored in `IN1` and `IN2` (`SMT.SMTPTServer(IN1,IN2)`). The result of `SMTPTServer` is kept in the variable `OUT1`. Finally, the state machine transforms to the final state with a result of service composition kept in `OUT1`.

Essentially, we have verified that the SPATEL is in a format as described in section 2.3.2. Indeed, this is not really a service composition as it is composed of only one service component. The user does not need a service composition as the already existed service in the service repository is sufficient to fulfill his request. Hence, we have shown in this example that IA is not only able to construct service compositions but also able to decide when the compositions are not necessary. If the compositions are not necessary, IA will simply propose a right service in the repository to the user (the candidate is still in SPATEL format).

Furthermore, the SPATEL represents the right service input parameter for the service. For example, the two inputs for `SMTPTServer` are received from the user as defined in the `ContextRule.txt`.

5.1.3.2 Result of scenario 3: Service input parameters

In this section, we present service input parameters extracted from the user request. For the service composition deduced from recommended service rule, the input parameters are obtained directly from the file (`ContextRule.txt`). Therefore, we are not interested in the input parameters for that case. The service input parameters extracted by IA (without MRS) from a user request: “Find an address of theater in my area and send it to my email” is shown in Figure 100.

The syntax is `<Service name, Service goal>, [service input, matching weight]`. For example, we can read the first entry as: an input for a service, `GlobalSMS`, that has a service goal, `SendSms`, is `Remco@telin.nl`. It has 80 percent (i.e. a matching weight is 8 out of 10) chance that `Remco@telin.nl` is an input of the service `GlobalSMS`. Notice that Remco@telin.nl is derived from a keyword “my email” in the user request. In this thesis, We assume that Remco@telin.nl is an email address of the end user.

Indeed, some of the input parameters are not accurate. For example, `Remco@telin.nl` should not be used as an input of `GlobalSMS` that receives a mobile phone address as an input (although a mobile phone address and an email address are semantically similar as defined in Input/Output ontology, the email address should not be used as the mobile phone address). This result can be improved by using MRS. As discussed in section 3.5.8, IA verifies the service input parameters matching with MRS of the sentence (i.e. user request). Basically, IA ensures that the services and their inputs in Figure 100 agree with a meaning of the sentence.

As shown in Figure 101, service input parameters that fail to agree with MRS have been removed. This makes the input parameters extraction becomes more accurate. It is worth noting that, as opposed to user scenario 1 and scenario 2, IA could extract service input for the service, `MailDaemon`, for example. This is because the user request is complete. IA could infer an

Result and Discussion

owner of the email from the sentence (i.e. the sentence does explicitly state the owner of the email via the phrase “my email”). This verifies our claim in section 5.1.1.2 and 5.1.2.2.

```
Inputs from user:
[<GlobalSMS,SendSms>,[<Remco@telin.nl,8>]][<UKLocation,GetLocation>,[<A2.05@telin.
nl,8>]][<URLToIPAddress,GetIpAddress>,[<Remco@telin.nl,8>]][<GetRestaurantByAddress
,FindRestaurantByAddress>,[<A2.05@telin.nl,8>]][<MessagingServer,SendSms>,[<Remco@t
elin.nl,8>]][<MessagingServer,SendInformation>,[<Remco@telin.nl,9>]][<MailDaemon,Send
Email>,[<Remco@telin.nl,9>]][<TheaterSearch,SearchTheaterByAddress>,[<A2.05@telin.nl,
8>]][<UKLocationPro,GetLocation>,[<A2.05@telin.nl,9>]]
```

Figure 100: Service input parameters extracted from a request “Find an address of theater in my area and send it to my email” (IA without MRS)

```
Inputs from user:
[<GetRestaurantByAd-
dress,FindRestaurantByAddress>,[<A2.05@telin.nl,8>]][<MailDaemon,SendEmail>,[<Remco
@telin.nl,9>]][<TheaterSearch,SearchTheaterByAddress>,[<A2.05@telin.nl,8>]]
```

Figure 101: Service input parameters extracted from a request “Find an address of theater in my area and send it to my email” (IA with MRS)

5.2 IA Correctness

In this section, we verify that the requirements (i.e. functionalities) presented in section 3.1 have been met. It is worth noting that although we have verified IA correctness in the three user scenarios, we will not explain the results separately. Instead of going into the requests of each user scenarios one by one, we demonstrate the IA correctness with general requests (provided in the subsequent paragraphs). This is because the explanations given to the general requests can be applied in general (to the requests of the three user scenario). Therefore, we decide not to duplicate the explanations. However, for detail explanations on the requests of each user scenario, we refer to pdf files located in *spatel* folder of this thesis (Appendix F).

The request used in this section is “I want to get traffic condition in Paris and send it to my email”.

Initially, we ensure that the natural language request is processed correctly. This will verify requirement (property) 3-9 (in section 3.1). We have already verified requirement 10 and 12 in section 5.1 (That is, we have verified that the service compositions were deployed in a correct SPATEL format).

Next, we select an option to supplement IA knowledge with a MRS result of NLP-tool (the MRS is shown in Appendix C). This will verify requirement 1 (i.e. IA provides an easy-to-use interface that the user could configure the IA features) and 11. The request used to test this feature is “Find traffic in Paris and send them by email”. This is a simplified version of, “I want to get traffic condition in Paris and send them by email”. The sentence corresponds to a Norwegian sentence of “finn trafikkkforholdene i Paris og send dem pr. Email”, in which the NLP-tool supports.

Next, we enable a context-handling feature. The context source is used to provide location context and authentication context to IA in order to test requirement 13-17 and 18-19, respectively.

Result and Discussion

We set up the recommended service rule and the restriction rule for each user context as shown in Figure 102 and Figure 103, respectively.

Finally, we verify that the context will not be processed by IA if the context belongs to other users (requirement 18-19 (section 3.1)). Hence, all requirements related to the correctness of IA functionalities will be verified. However, IA performance (i.e. requirement 2) will be measured (i.e. verified) during the prototype demonstration (section 5.4) based on methods discussed in section 5.3.

IA correctness verifications are shown in the subsequent paragraphs:

```
! context rule, invoke operation of service when the context occurs
!
! Context          Service          operation          input parameter
! -----
"A2.05@telin.nl"  "CITYTrafficReport" "ReportTrafficCondition" "Paris"
"A2.05@telin.nl"  "MessagingServer"   "SmsSender"        "TrafficRe-
port,00475612605"
"Canteen@telin.nl" "GlobalNews"        "SendNews"         "00475612605"
```

Figure 102: IA Recommended Service Rule

```
! restriction rule, restrict an access to service when the context occurs
!
! Context          Service
! -----
"A2.07@telin.nl"  "MailDaemon,EmailDelivery"
```

Figure 103: IA Restriction Service Rule

Results obtained from an execution of IA with a user request, “I want to get traffic condition in Paris and send it to my email”, are illustrated below:

- All keywords are extracted from goal ontology. This is shown in Figure 104. Hence, we verified requirement 3

```
All keywords= [Book, Sas, Flight, Restaurant, Sterling, Find, By, Location, And, Type, Mp3, Address, Of, Forecast, In, France, Get, Price, Info, Nybot, Email, Geographic, Country, Distance, Currency, Exchange, Rate, News, Iso, Code, Financial, Kcbot, Goals, Ip, Holiday, Dates, Airport, Stock, Recipe, Joke, Business, Company, Name, User, Information, U, S, Zip, Area, Job, Scotland, Residence, Lme, Fedex, Shipping, Web, Ranking, Tree, City, State, Cbot, Cmeq, North, Ireland, Geo, Mgex, Comex, Gb, Phone, Number, Digital, Signature, Quote, Api, Uk, Interest, Wce, Traffic, Condition, Us, Treasury, Uri, Map, Nymex, Provide, Voice, Report, Search, Norwegian, Send, Fax, Icq, Text, Theater, Langauge, Vegetarian, For, Bank, Start, Time, Sms, Chinese, Mms, Translate, To, French, Hungarian, Spanish, English, Weather]
```

Figure 104: All keywords found in goal ontology

- The semantic-distance table is constructed. This is shown in Figure 105. For example, a semantic distance between a service goal, BookSasFlight and another service goal, BookFlight is 1. This means that these two goals are closely related. Hence, we verified requirement 5

Result and Discussion

```
Build Table ...
ontclass1 = http://www.spicerepository.net/Goals.owl#BookSasFlight ontclass2=
http://www.spicerepository.net/Goals.owl#BookFlight
distance = 1
ontclass1 = http://www.spicerepository.net/Goals.owl#BookSasFlight ontclass2=
http://www.spicerepository.net/Goals.owl#Book
distance = 2
ontclass1 = http://www.spicerepository.net/Goals.owl#BookSasFlight ontclass2=
http://www.spicerepository.net/Goals.owl#BookRestaurant
distance = 3
ontclass1 = http://www.spicerepository.net/Goals.owl#BookSasFlight ontclass2=
http://www.spicerepository.net/Goals.owl#BookSterlingFlight
distance = 2
ontclass1 = http://www.spicerepository.net/Goals.owl#BookSasFlight ontclass2=
http://www.spicerepository.net/Goals.owl#BookFlight
distance = 1
ontclass1 = http://www.spicerepository.net/Goals.owl#BookSasFlight ontclass2=
http://www.spicerepository.net/Goals.owl#BookSterlingFlight
```

Figure 105: IA builds semantic-distance table

- All keywords found in each service goal are extracted. For example, GetMap consists of two keywords. The two keywords are “Get” and “Map”. This is shown in Figure 106. Hence, we verified requirement 5

```
goal = GetMap
keywords= [Get, Map]
goal = GetPriceInfoInNymex
keywords= [Get, Price, Info, In, Nymex]
goal = Get
keywords= [Get]
goal = ProvideNews
keywords= [Provide, News]
goal = ProvideVoiceApi
keywords= [Provide, Voice, Api]
goal = Report
keywords= [Report]
goal = SearchNorwegianRestaurant
keywords= [Search, Norwegian, Restaurant]
```

Figure 106: IA extracts keywords from service goals

- A weight for each keyword in service goals is calculated. This is shown in Figure 107. For example, a keyword “Stock” has a weight equal to 100. These weights are properly combined to get a maximum weight for each service goal. These weights are also used to evaluate how well the input keywords are linked with the service goals. Hence, we verified requirement 5

```
ptable= {Goals=100, Stock=100, Theater=100, France=100, Sms=50, City=16, Book=33, Map=100, Signa-
ture=100, Joke=100, Norwegian=20, Interest=100, Geographic=50, U=16, State=16, Search=25, Traffic=100,
Iso=100, Weather=50, Uk=100, Phone=25, Info=14, Scotland=100, English=100, Cbot=100, Tree=100, Icq=100,
Area=100, Hungarish=100, Type=100, Time=100, Distance=100, By=12, Forecast=50, Ireland=100, Flight=50,
To=50, Recipe=100, Uri=100, Spanish=100, Mp3=100, Sterling=100, Dates=50, Us=16, Address=25, Code=20,
Get=14, In=12, Name=14, Chinese=100, Wce=100, Mgex=100, Business=100, Ip=33, User=100, Langauge=100,
Kcvt=100, Comex=100, Exchange=100, Number=25, For=33, Sas=100, Rate=50, Bank=100, Send=50,
French=100, Api=100, Text=50, Financial=100, Fax=100, Provide=25, And=12, Fedex=100, S=14, Ranking=100,
Report=33, Lme=100, Cmeq=100, Quote=100, Gb=100, Nymex=100, Restaurant=16, Residence=100, Start=100,
Condition=100, Country=20, Company=25, Geo=20, North=100, News=100, Voice=100, Information=20, Vege-
tarian=100, Currency=100, Job=100, Of=100, Shipping=100, Digital=100, Web=100, Zip=16, Find=20,
```


Result and Discussion

Email=25, Price=12, Translate=50, Treasury=100, Holiday=50, Airport=100, Nybot=100, Mms=100, Location=14}

Figure 107: IA calculates a weight for each keyword

- Keywords (e.g. of type noun, verb) in the user request are matched with service goals. This is shown in Figure 108. For example, there is one keyword (e.g. “Get”) matches with a service goal, GetIsoCode. Hence, we verified requirement 3 and 4

```
++Keyword Match++
key = GetIsoCode value = [Get]
key = GetMap value = [Get]
key = GetBusinessInfoByPhoneNumber value = [Get]
key = FindRestaurantByLocationAndType value = [Find]
key = GetQuote value = [Get]
key = GetDistance value = [Get]
key = GetPriceInfoInMgex value = [Get]
key = SendText value = [Send]
key = GetStockInfo value = [Get]
key = GetWebRanking value = [Get]
key = SearchTheaterByStartTime value = [Search]
key = SearchRestaurantByName value = [Search]
key = FindMp3 value = [Find]
key = FindAddressOfLocation value = [Find]
key = GetGeographicInfoByCity value = [Get]
key = GetEmail value = [Get, Email]
key = SearchTheaterByCityName value = [Search]
key = GetJobInfo value = [Get]
key = GetLocationByIp value = [Get]
key = SearchTheaterByStateName value = [Search]
key = GetHolidayDatesInGb value = [Get]
key = GetRecipe value = [Get]
key = GetAreaCode value = [Get]
key = SearchTheaterByLanguage value = [Search]
key = GetPriceInfoInNymex value = [Get]
key = FindRestaurantByLocation value = [Find]
key = GetFinancialInfoByUri value = [Get]
key = GetJoke value = [Get]
key = GetCurrencyExchangeRate value = [Get]
key = GetTrafficCondition value = [Get, Traffic, Condition]
key = GetPriceInfoInComex value = [Get]
key = SearchRestaurantByType value = [Search]
key = SendSms value = [Send]
key = SendInformation value = [Send]
key = SendEmail value = [Send, Email]
key = GetGeographicInfoByCountry value = [Get]
key = GetHolidayDatesInScotland value = [Get]
key = GetCompanyInfoByName value = [Get]
key = GetFedexShippingRate value = [Get]
key = GetInterestInfo value = [Get]
key = GetHolidayDatesByCity value = [Get]
key = SearchForBankCode value = [Search]
key = GetLocationByZipCode value = [Get]
key = SearchFlight value = [Search]
key = SearchRestaurantByPrice value = [Search]
key = GetPriceInfoInKcvt value = [Get]
key = SearchTheaterByPrice value = [Search]
key = SearchNorwegianRestaurant value = [Search]
key = SearchTheaterByZipCode value = [Search]
key = SendMms value = [Send]
```

Result and Discussion

```
key = SendIcqText value = [Send]
key = GetFinancialInfoByCompanyName value = [Get]
key = GetIpAddress value = [Get]
key = SearchRestaurantByZipCode value = [Search]
key = SearchVegetarianRestaurant value = [Search]
key = FindRestaurantByAddress value = [Find]
key = GetPriceInfoInNybot value = [Get]
key = GetPriceInfoInLme value = [Get]
key = GetBusinessTree value = [Get]
key = GetHolidayDatesInNorthIreland value = [Get]
key = SearchChineseRestaurant value = [Search]
key = GetAirportInfo value = [Get]
key = GetResidenceInfoByPhoneNumber value = [Get]
key = GetCityNameAndStateNameByZipCode value = [Get]
key = FindRestaurantInFrance value = [Find]
key = GetPriceInfoInWce value = [Get]
key = GetGeoDistance value = [Get]
key = GetPriceInfoInCmeq value = [Get]
key = SendSmsText value = [Send]
key = SearchTheWeb value = [Search]
key = GetFinancialInfoInUSTreasury value = [Get]
key = SendNews value = [Send]
key = SendFax value = [Send]
key = GetPositionByUsZipCode value = [Get]
key = GetPriceInfoInCbot value = [Get]
key = GetLocation value = [Get]
key = FindCellIdLocation value = [Find]
key = GetUserInformation value = [Get]
key = GetDigitalSignature value = [Get]
```

Figure 108: IA matches input keywords with service goals

- A list of consolidated services is constructed. This is shown in Figure 109. The first entry in Figure 109 can be interpreted as follows. A service that has a service goal, SendIcqText has a maximum goal weight equals to 200 ($acc = 200$). A threshold, which is a criteria to select the service as consolidated service, is equal to 100 ($acc/2 = 100$). As a matching scale (a sum of keywords, in the user request, that are matched with this service goal) is less than threshold (matching scale = 50). The service has not been selected as a consolidated service (consolidated goal = []). On the other hand, a service that has a service goal, SendEmail, is selected as a consolidated service (consolidated goal = [SendEmail]). This is because the service has a matching scale more than or equal to the threshold (threshold = 37, matching scale = 75). Essentially, services that have weight less than corresponding threshold are filtered out. Hence, we verified requirement 4 and 6

```
goal = SendIcqText
keywords= [Send, Icq, Text]
acc = 200
matching scale = 50
threshold = 100
consolidated goal = []
goal = SendEmail
keywords= [Send, Email]
acc = 75
matching scale = 75
threshold = 37
consolidated goal = [SendEmail]
```

Figure 109: IA constructs a list of consolidated services (corresponding to service goals)

Result and Discussion

- Instances of service input parameters (getting from the user request) are mapped to corresponding services. This is shown in Figure 110. For example, an input of a service, CITYTrafficReport, that has a service goal, GetTrafficCondition, is Paris. It has 90 percent (9 out of 10) chance that Paris is an input of the service (<Paris,9>). Hence, we verified requirement 9

```
<service.goal> = <CITYTrafficReport,GetTrafficCondition>
<mapWithInput>
<Paris,9>
z= [<Paris,9>]
</mapWithInput>
```

Figure 110: IA maps instances of service input parameters to corresponding services

- A list of composition candidates is constructed from services in the list of consolidated services. This is shown in Figure 111. A syntax of the result is described in section 5.4.1. Hence, we verified requirement 7

```
++List of Composition Candidates++
[<CITYTrafficReport,0>]
[<StreetTrafficReport,0>]
[<USTrafficReport,0>]
[<MessagingServer,0>, <MailLocate,20>]
[<MailLocate,0>]
[<MessagingServer,0>, <CITYTrafficReport,50>]
[<MessagingServer,0>, <StreetTrafficReport,50>]
[<MessagingServer,0>, <USTrafficReport,50>]
[<MessagingServer,0>, <UKLocation,50>, <CITYTrafficReport,14>]
[<MessagingServer,0>, <UKLocation,50>, <StreetTrafficReport,14>]
[<MessagingServer,0>, <UKLocation,50>, <USTrafficReport,14>]
[<MessagingServer,0>, <UKLocationPro,50>]
[<CDYNESMSNotify,0>, <CITYTrafficReport,50>]
[<CDYNESMSNotify,0>, <StreetTrafficReport,50>]
[<CDYNESMSNotify,0>, <USTrafficReport,50>]
[<CDYNESMSNotify,0>, <UKLocation,50>, <CITYTrafficReport,14>]
[<CDYNESMSNotify,0>, <UKLocation,50>, <StreetTrafficReport,14>]
[<CDYNESMSNotify,0>, <UKLocation,50>, <USTrafficReport,14>]
[<CDYNESMSNotify,0>, <UKLocationPro,50>]
[<GlobalSMS,0>, <CITYTrafficReport,50>]
[<GlobalSMS,0>, <StreetTrafficReport,50>]
[<GlobalSMS,0>, <USTrafficReport,50>]
[<GlobalSMS,0>, <MailLocate,5>]
[<GlobalSMS,0>, <UKLocation,50>, <CITYTrafficReport,14>]
[<GlobalSMS,0>, <UKLocation,50>, <StreetTrafficReport,14>]
[<GlobalSMS,0>, <UKLocation,50>, <USTrafficReport,14>]
[<GlobalSMS,0>, <UKLocationPro,50>]
[<MailDaemon,0>, <CITYTrafficReport,50>]
[<MailDaemon,0>, <StreetTrafficReport,50>]
[<MailDaemon,0>, <USTrafficReport,50>]
[<MailDaemon,0>, <MailLocate,20>]
[<MailDaemon,0>, <UKLocation,50>, <CITYTrafficReport,14>]
[<MailDaemon,0>, <UKLocation,50>, <StreetTrafficReport,14>]
[<MailDaemon,0>, <UKLocation,50>, <USTrafficReport,14>]
[<MailDaemon,0>, <UKLocationPro,50>]
[<UKLocation,0>, <CITYTrafficReport,14>]
[<UKLocation,0>, <StreetTrafficReport,14>]
[<UKLocation,0>, <USTrafficReport,14>]
```

Result and Discussion

```
[<UKLocationPro,0>]
end ++List of Composition Candidates++
```

Figure 111: IA constructs a list of service composition candidates

- The composition candidates are sorted according to their weight. This is shown in Figure 112. A syntax of the result is described in section 5.4.1. Hence, we verified requirement 8

```
++List of Composition Candidates sorted by total weight++
[<MailDaemon,0>, <UKLocation,50>, <USTrafficReport,14>]
[<MailDaemon,0>, <UKLocation,50>, <CITYTrafficReport,14>]
[<MailDaemon,0>, <UKLocation,50>, <StreetTrafficReport,14>]
[<MessagingServer,0>, <UKLocation,50>, <StreetTrafficReport,14>]
[<MessagingServer,0>, <UKLocation,50>, <USTrafficReport,14>]
[<CDYNESMSNotify,0>, <UKLocation,50>, <USTrafficReport,14>]
[<GlobalSMS,0>, <UKLocation,50>, <StreetTrafficReport,14>]
[<GlobalSMS,0>, <UKLocation,50>, <USTrafficReport,14>]
[<GlobalSMS,0>, <UKLocation,50>, <CITYTrafficReport,14>]
[<CDYNESMSNotify,0>, <UKLocation,50>, <CITYTrafficReport,14>]
[<CDYNESMSNotify,0>, <UKLocation,50>, <StreetTrafficReport,14>]
[<MessagingServer,0>, <UKLocation,50>, <CITYTrafficReport,14>]
[<MailDaemon,0>, <CITYTrafficReport,50>]
[<MailDaemon,0>, <StreetTrafficReport,50>]
[<MailDaemon,0>, <USTrafficReport,50>]
[<CDYNESMSNotify,0>, <CITYTrafficReport,50>]
[<CDYNESMSNotify,0>, <StreetTrafficReport,50>]
[<CDYNESMSNotify,0>, <USTrafficReport,50>]
[<GlobalSMS,0>, <USTrafficReport,50>]
[<GlobalSMS,0>, <CITYTrafficReport,50>]
[<MessagingServer,0>, <CITYTrafficReport,50>]
[<MessagingServer,0>, <StreetTrafficReport,50>]
[<MessagingServer,0>, <USTrafficReport,50>]
[<GlobalSMS,0>, <StreetTrafficReport,50>]
[<UKLocation,0>, <StreetTrafficReport,14>]
[<UKLocation,0>, <USTrafficReport,14>]
[<UKLocation,0>, <CITYTrafficReport,14>]
[<CITYTrafficReport,0>]
[<StreetTrafficReport,0>]
[<USTrafficReport,0>]
[<MailDaemon,0>, <UKLocationPro,50>]
[<MailDaemon,0>, <MailLocate,20>]
[<CDYNESMSNotify,0>, <UKLocationPro,50>]
[<GlobalSMS,0>, <UKLocationPro,50>]
[<MessagingServer,0>, <UKLocationPro,50>]
[<MessagingServer,0>, <MailLocate,20>]
[<GlobalSMS,0>, <MailLocate,5>]
[<MailLocate,0>]
[<UKLocationPro,0>]
end ++List of Composition Candidates sorted by total weight++
```

Figure 112: IA ranks a list of service composition candidates

Results obtained from an execution of IA (that is supplement with knowledge of NLP-tool) with a user request, “Find Traffic in Paris and send them by email”, are illustrated below. As mentioned earlier, NLP-tool does not support analysis in English. Hence, the equivalence statement in Norwegian is “finn trafikkkforholdene i Paris og send dem pr. Email”. Furthermore, we have discussed that, for a simple sentence, a dictionary file can be used to translate the MRS result in Norwegian to the one in English. The translated MRS is then used by IA to im-

Result and Discussion

prove its accuracy while it processes the request. There are two improvements obtained from MRS. The first improvement is on a ranking of the service composition candidates. The second improvement is on a matching of service input parameters. As we have verified and discussed on the second improvement in section 5.1, we would concentrate on the first improvement at this point.

- The composition candidates are extracted and sorted according to their weight. This is shown in Figure 113. A syntax of the result is described in section 5.4.1. Hence, we verified requirement 8

```
++List of Composition Candidates sorted by total weight++
[<MailDaemon,0>, <UKLocation,50>, <USTrafficReport,14>]
[<MailDaemon,0>, <UKLocation,50>, <CITYTrafficReport,14>]
[<MailDaemon,0>, <UKLocation,50>, <StreetTrafficReport,14>]
[<MessagingServer,0>, <UKLocation,50>, <StreetTrafficReport,14>]
[<MessagingServer,0>, <UKLocation,50>, <USTrafficReport,14>]
[<CDYNESMSNotify,0>, <UKLocation,50>, <USTrafficReport,14>]
[<GlobalSMS,0>, <UKLocation,50>, <StreetTrafficReport,14>]
[<GlobalSMS,0>, <UKLocation,50>, <USTrafficReport,14>]
[<GlobalSMS,0>, <UKLocation,50>, <CITYTrafficReport,14>]
[<CDYNESMSNotify,0>, <UKLocation,50>, <CITYTrafficReport,14>]
[<CDYNESMSNotify,0>, <UKLocation,50>, <StreetTrafficReport,14>]
[<MessagingServer,0>, <UKLocation,50>, <CITYTrafficReport,14>]
[<MailDaemon,0>, <CITYTrafficReport,50>]
[<MailDaemon,0>, <StreetTrafficReport,50>]
[<MailDaemon,0>, <USTrafficReport,50>]
[<CDYNESMSNotify,0>, <CITYTrafficReport,50>]
[<CDYNESMSNotify,0>, <StreetTrafficReport,50>]
[<CDYNESMSNotify,0>, <USTrafficReport,50>]
[<GlobalSMS,0>, <USTrafficReport,50>]
[<GlobalSMS,0>, <CITYTrafficReport,50>]
[<MessagingServer,0>, <CITYTrafficReport,50>]
[<MessagingServer,0>, <StreetTrafficReport,50>]
[<MessagingServer,0>, <USTrafficReport,50>]
[<GlobalSMS,0>, <StreetTrafficReport,50>]
[<UKLocation,0>, <StreetTrafficReport,14>]
[<UKLocation,0>, <USTrafficReport,14>]
[<UKLocation,0>, <CITYTrafficReport,14>]
[<MailDaemon,0>, <UKLocationPro,50>]
[<MailDaemon,0>, <MailLocate,20>]
[<CITYTrafficReport,0>]
[<StreetTrafficReport,0>]
[<USTrafficReport,0>]
[<CDYNESMSNotify,0>, <UKLocationPro,50>]
[<GlobalSMS,0>, <UKLocationPro,50>]
[<MessagingServer,0>, <UKLocationPro,50>]
[<MessagingServer,0>, <MailLocate,20>]
[<GlobalSMS,0>, <MailLocate,5>]
[<MailLocate,0>]
[<SkyScanner,0>]
[<UKLocationPro,0>]
end ++List of Composition Candidates sorted by total weight++
```

Figure 113: IA ranks a list of service composition candidates (2)

- The sorted list of composition candidates are validated by the MRS result of NLP-tool. The candidates that are not agreed with the MRS result will be moved to the end of the list. Therefore, the ranking of service composition candidates is more accurate (than the

Result and Discussion

ranking without MRS). This is shown in Figure 113. A syntax of the result is described in section 5.4.1. Hence, we verified requirement 11

```
++List of Composition Candidates sorted by nlp++
[<MailDaemon,0>, <CITYTrafficReport,50>]
[<MailDaemon,0>, <StreetTrafficReport,50>]
[<MailDaemon,0>, <USTrafficReport,50>]
[<CDYNESMSNotify,0>, <CITYTrafficReport,50>]
[<CDYNESMSNotify,0>, <StreetTrafficReport,50>]
[<CDYNESMSNotify,0>, <USTrafficReport,50>]
[<GlobalSMS,0>, <USTrafficReport,50>]
[<GlobalSMS,0>, <CITYTrafficReport,50>]
[<MessagingServer,0>, <CITYTrafficReport,50>]
[<MessagingServer,0>, <StreetTrafficReport,50>]
[<MessagingServer,0>, <USTrafficReport,50>]
[<GlobalSMS,0>, <StreetTrafficReport,50>]
[<UKLocation,0>, <StreetTrafficReport,14>]
[<UKLocation,0>, <USTrafficReport,14>]
[<UKLocation,0>, <CITYTrafficReport,14>]
[<MailDaemon,0>, <UKLocation,50>, <USTrafficReport,14>]
[<MailDaemon,0>, <UKLocation,50>, <CITYTrafficReport,14>]
[<MailDaemon,0>, <UKLocation,50>, <StreetTrafficReport,14>]
[<MessagingServer,0>, <UKLocation,50>, <StreetTrafficReport,14>]
[<MessagingServer,0>, <UKLocation,50>, <USTrafficReport,14>]
[<CDYNESMSNotify,0>, <UKLocation,50>, <USTrafficReport,14>]
[<GlobalSMS,0>, <UKLocation,50>, <StreetTrafficReport,14>]
[<GlobalSMS,0>, <UKLocation,50>, <USTrafficReport,14>]
[<GlobalSMS,0>, <UKLocation,50>, <CITYTrafficReport,14>]
[<CDYNESMSNotify,0>, <UKLocation,50>, <CITYTrafficReport,14>]
[<CDYNESMSNotify,0>, <UKLocation,50>, <StreetTrafficReport,14>]
[<MessagingServer,0>, <UKLocation,50>, <CITYTrafficReport,14>]
[<MailDaemon,0>, <UKLocationPro,50>]
[<MailDaemon,0>, <MailLocate,20>]
[<CITYTrafficReport,0>]
[<StreetTrafficReport,0>]
[<USTrafficReport,0>]
[<CDYNESMSNotify,0>, <UKLocationPro,50>]
[<GlobalSMS,0>, <UKLocationPro,50>]
[<MessagingServer,0>, <UKLocationPro,50>]
[<MessagingServer,0>, <MailLocate,20>]
[<GlobalSMS,0>, <MailLocate,5>]
[<MailLocate,0>]
[<SkyScanner,0>]
[<UKLocationPro,0>]
end ++List of Composition Candidates sorted by nlp++
```

Figure 114: IA ranks a list of service composition candidates based on MRS

Result obtained from IA when the user location has been changed to, “A2.05@telin.nl”, is illustrated below:

- When the user location is changed to [A2.05@telin.nl](#), the service composition as defined in ContextRule.txt is invoked. This is shown in Figure 115. It is illustrated that a service composition of two service components (as defined in the ContextRule.txt), SendSms and GetTrafficCondition, is generated. Hence, we verified requirement 12-17

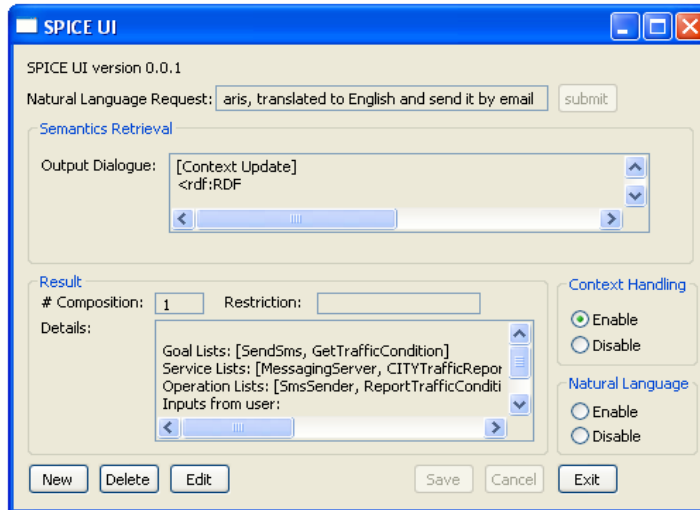


Figure 115: IA invokes a service composition defined in ContextRule.txt

Result obtained from IA when the user location has been changed to, “A2.07@telin.nl”, is illustrated below:

- When the user location is changed to A2.07@telin.nl, the service MailDaemon and EmailDelivery as defined in RestrictionRule.txt is restricted. As a result, number of service composition satisfied the user request (“I want to get traffic condition in Paris and send it to me by email or send it to me by sms”) is reduced from 47 (not shown) to 40. This is shown in Figure 116. Hence, we verified requirement 12-17



Figure 116: IA restricts access to services as defined in RestrictionRule.txt

Result obtained from IA, when it receives a change in restriction context, is illustrated below:

- Up on receiving the restriction context on *GlobalSMS*, an access to *GlobalSMS* has been restricted. As a result, number of service composition satisfied the user request (“I

Result and Discussion

want to get traffic condition in Paris and send it to me by email or send it to me by sms”) is reduced from 40 (in Figure 116) to 30. This is shown in Figure 117. Hence, we verified requirement 19-20

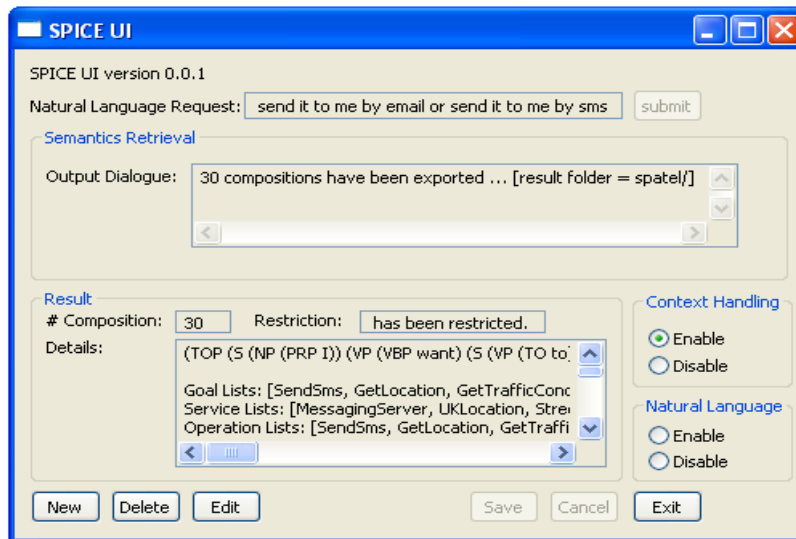


Figure 117: IA restricts an access to service, GlobalSMS as response to a change in restriction context

Result obtained from IA, when it receives a change in authentication context, is illustrated below:

- Up on receiving the authentication context on *GlobalSMS*, an access to *GlobalSMS* has been authenticated. As a result, number of service composition satisfied the user request (“I want to get traffic condition in Paris and send it to me by email or send it to me by sms”) is increased from 30 to 40. This is shown in Figure 118. Hence, we verified requirement 19-20

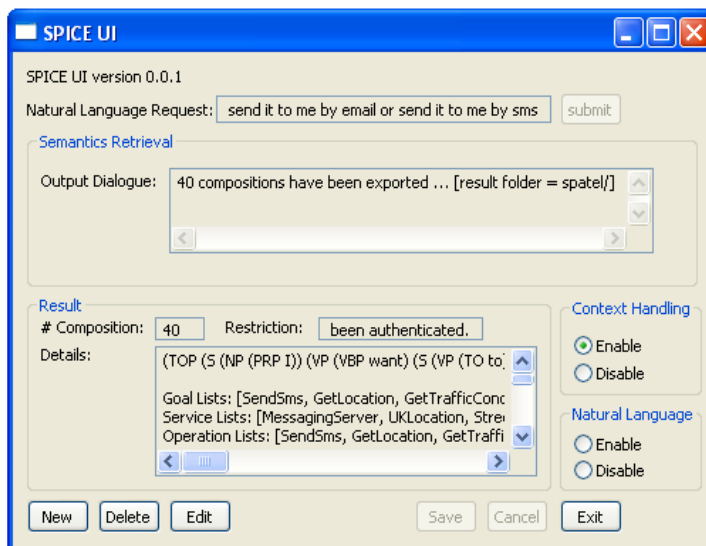


Figure 118: IA allows an access to service, GlobalSMS as response to a change in authentication context

Result and Discussion

The user context discussed earlier is a context of the user, Remco.Poortinga@telin.nl (i.e. Remco.Poortinga@telin.nl is the IA user). To verify that IA does not process a user context if the context belongs to other users, the context source is then used to publish a change in user location to A2.05@telin.nl (Figure 119). This context belongs to other users, such as Marin.Wibbels@telin.nl. Since this context belongs to other users, IA ignores the context and does not compose for any services. The result is shown in Figure 120. The number of service composition is 0. This is in contrast to Figure 115 where one service has been composed.

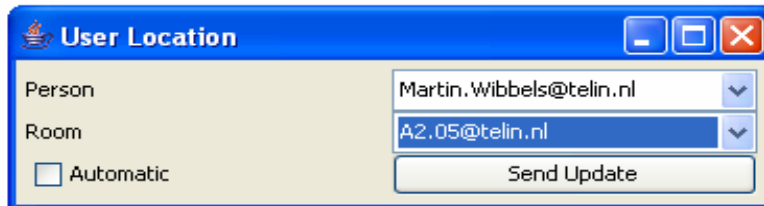


Figure 119: A user location is changed to A2.05@telin.nl for user, Martin.Wibbels@telin.nl

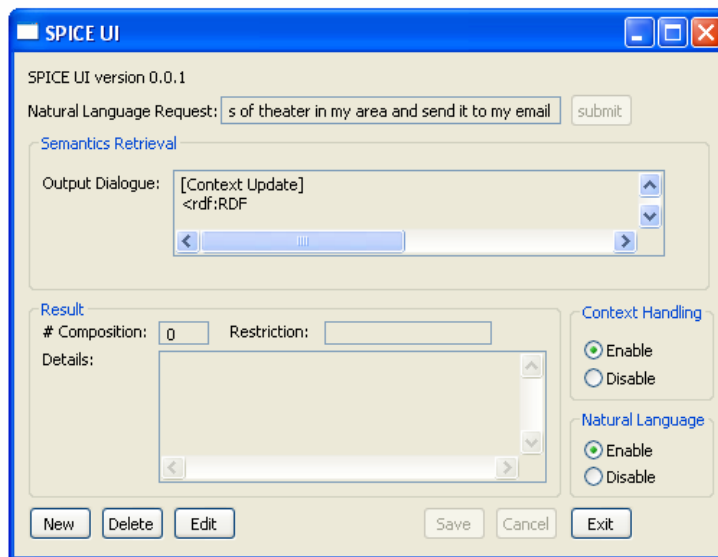


Figure 120: IA ignores the context that belongs to different users

If a service takes an address as its input parameter, the address could be derived from the context source providing that the context handling feature is enabled. To verify that IA is able to obtain an instance of input parameters from the user context, the context source is used to publish a change in user location to A2.06@telin.nl.

A mapping between service and its input parameter (for the request “Find vegetarian restaurant in my area and send it by sms or email”) is shown in Figure 121. Figure 121 is interpreted as follows: A service VEGRestaurantFinding that has a goal SearchVegetarianRestaurant takes a service input instance A2.06@telin.nl. This input instance, A2.06@telin.nl, has an appropriate level equals to 8 (out of 10). In other words, 8 indicates how well A2.06@telin.nl (e.g. user location) is suitable to be used as input instance of VEGRestaurantFinding that takes address as its input parameter. Other elements (e.g. [`<UKLocation,GetLocation>`,`[A2.06@telin.nl,8>`]]) in Figure 121 are interpreted in the same way. As expected, the input for VEGRestaurantFinding is A2.06@telin.nl. If the context handling is not enabled, the input for VEGRestaurantFinding is A2.05@telin.nl which is a default location.

```
Inputs from
user[<VEGRestaurantFinding,SearchVegetarianRestaurant>,[<A2.06@telin.nl,8>]][<UKLocation,GetLocation>,[<A2.06@telin.nl,8>]][<GetRestaurantByAddress,FindRestaurantByAddress>,[<A2.06@telin.nl,8>]][<GetRestaurantByCoords,FindRestaurantByLocation>,[<A2.06@telin.nl,8>]][<UKLocationPro,GetLocation>,[<A2.06@telin.nl,9>]]
```

Figure 121: A mapping between service and its input parameters derived from user context

5.3 Performance Measurement

We measure the performance of IA from two aspects. The two main aspects are query quality and query response time [50]. Precision and Recall [50] are the major factors that affect query quality. They measure relativity between search results and user request.

Specifically, precision is defined as the ratio of the number of relevant services retrieved to the whole retrieved number. Recall is defined as the ratio of the relevant services retrieved number to the number of all relevant services in system. We define relevant services as 1) the services that can satisfy the user request or 2) the services that can be composed with other services to satisfy the user request.

In this thesis, there are two ways in perceiving service. Firstly, the service could be a service offered by a particular service provider (i.e. an entry in the service repository). Secondly, the service could be composed from other services (i.e. it is composed from other services that are entries of the service repository).

Therefore, two precision indexes are used to measure query quality (one precision index is used to measure individual service and another precision index is used to measure service composition). However, only one “recall” is used (i.e. it is used to measure individual service). This is because recall for service compositions is hard to measure. Giving a large number of services in the service repository, there could be a large number of all relevant services in system. In other words, the number of services that can be obtained from service composition is huge.

If we have N service in the service repository and suppose that every service could be linked with each other, the number of service composition is $N!$ (e.g. the first service can link with other $n-1$ services, the second service can link with $n-2$ service and so on). Furthermore, it is difficult to filter out the relevant services from a large number of services. Hence, for simplicity, we will base our query-quality measurement on two precision indexes and one recall.

As for a query response time, there are two major factors that affect the response time. These are the total number of services in system and the number of service input/output parameters. Apparently, if the total number of services is large, IA requires more time to match keywords with corresponding service goals. Moreover, if the services take many input parameters and produce many outputs, IA requires more time to compose services (i.e. IA has to check semantic similarity of all inputs and outputs of services). However, IA speeds up the process of checking semantic similarity of service inputs/outputs. This is done by statically builds up the semantic table (section 3.5.2). The semantic distance from the table will be retrieved by IA to find a semantic relationship between services input/output instead of calculating the semantic distance dynamically (for each user request).

We obtain a query response time by embedding a java object, *StopWatch* (defined in appendix G, the source code has been obtained from <http://www.javapractices.com/topic/TopicAction.do?Id=85>) in IA source code. *StopWatch* provides three simple methods, *start()*, *stop()* and *getTimedValue()* to start measuring the time, stop the watch and obtain the result (i.e. the time execution the code between *start()* and *stop()* method), respectively. It is worth noting that the time obtained from *StopWatch* is not precise. This is because it includes the time that is used to execute the *StopWatch* itself. However, it is sufficient for our demonstration purpose.

The query response time is calculated from the time that IA retrieves a request from the user until the time that the service composition candidates are derived. We do not include a time that is used to deploy the service composition candidates in SPATEL. This is because:

- IA depends on a module of ACE to deploy the service composition candidates (i.e. IA does not have a full control on the deployment process)
- The deployment process is not a part of service discovery. Although it is a part of service composition, it is not a mandatory part (e.g. the service composition candidates can be deployed in BPEL rather than SPATEL)

Results of IA performance based on these approaches are presented in subsequent sections.

5.4 Discussion on performance: User Scenarios

We have elaborated on the correctness of IA functionalities in section 5.2. Although we have also verified the correctness of IA functionalities (e.g. whether a list of consolidated services is correctly constructed, whether keywords from the request are correctly matched with the service goals) for the requests of the three user scenarios (the result files can be found in a *spatel* and *spatel2* folder (Appendix F) in a software package attached with this thesis), we will not discuss it in this section. This is because the discussion in section 5.2 is applied in general. In other words, the IA operations are the same (e.g. it has to build up the semantic table) regardless of the user request. Furthermore, we have verified functionalities of IA for every type of user contexts that IA supports in section 5.2. Hence, rather than, concentrating on the correctness of IA functionalities, we deliberate on IA performance using metrics introduced in section 5.3.

5.4.1 Format of the result

Test results are shown in Figure 122 - Figure 131. The format of the results can be described as follows:

The composition candidates are in the format of [*<service_1,weight_1>*, *<service_2,weight_2>*, ..., *<service_N,weight_N>*]. The weights (e.g. *weight_1*, *weight_2*, *weight_3*) show a degree in which the consecutive services (i.e. service N and service N-1) can be linked together. As the first service (*service_1*) in the composition does not associate with other service, *weight_1* always be zero. Furthermore, *weight_2* indicates how well *service_2* connects with *service_1*.

The composition candidates are ranked based on how well they are suitable for the user request. For example, in Figure 122, the first composition candidate is [*<SmsSender,0>*, *<UKLocation,50>*, *<VEGRestaurantFinding,8>*]. This composition is composed of three service components.

These service components are SmsSender, UKLocation and VEGRestaurantFinding. UKLocation and SmsSender can be connected together with a connection weight of 50. VEGRestaurantFinding and UKLocation can be connected together with a connect weight of 8.

5.4.2 Discussion on performance: User Scenario 1

As discussed in section 5.1.1, a user submits a request “Find vegetarian restaurant in my area and send it by sms or email”. IA processes a request in two alternative ways. Firstly, IA processes a request by using only the keyword search and ontology relation (the MRS functionality is disable). The composition candidates are shown in Figure 122.

These composition candidates are exported into SPATEL [43] which are ready to be deployed by SPATEL execution engine (in practice, these composition candidates should be proposed to the end user, and the end user selects the one that is most appropriate to his need. Hence, there should be only one composition candidate that will be exported and executed).

Precision index (for service individual) of the list of composition candidates is equal to 71.42 %. This means that out of 14 services retrieved (*GlobalSMS*, *VEGRestaurantFinding*, *UKLocationPro*, *SmsSender*, *UKLocation*, *MailDaemon*, *MessagingServer*, *PRICERestaurantFinding*, *RestaurantFinding*, *NORRestaurantFinding*, *GetRestaurantByAddress*, *GetRestaurantByCoords*, *MailLocate*, *SkyScanner*), there are four services unrelated to the request. The four services are *UKLocationPro*, *UKLocation*, *MailLocate* and *SkyScanner*. *UKLocation* and *UKLocationPro* are two services related to find a location of the user in UK. *MailLocate* is a service to find an email address of the particular user. *SkyScanner* is a service to search for a flight.

Thus, this give a precision index (for service individual) of $(14 - 4)/14$ which is 71.42 %. This indicates a higher performance than most traditional keyword-based ways of service discovery (for this particular scenario) [50]. Precision index (for service composition) of the list of composition candidates is equal to 16 %. This means that out of 100 service compositions (Figure 122), there are only sixteen compositions relevant to the request. The sixteen compositions are:

```
[<MessagingServer,0>, <VEGRestaurantFinding,33>]
[<GlobalSMS,0>, <VEGRestaurantFinding,33>]
[<SmsSender,0>, <VEGRestaurantFinding,33>]
[<MailDaemon,0>, <VEGRestaurantFinding,33>]
[<GlobalSMS,0>, <PRICERestaurantFinding,50>]
[<MessagingServer,0>, <PRICERestaurantFinding,50>]
[<SmsSender,0>, <PRICERestaurantFinding,50>]
[<MessagingServer,0>, <RestaurantFinding,33>]
[<GlobalSMS,0>, <RestaurantFinding,33>]
[<SmsSender,0>, <RestaurantFinding,33>]
[<SmsSender,0>, <NORRestaurantFinding,33>]
[<MessagingServer,0>, <NORRestaurantFinding,33>]
[<GlobalSMS,0>, <NORRestaurantFinding,33>]
[<MailDaemon,0>, <PRICERestaurantFinding,50>]
[<MailDaemon,0>, <RestaurantFinding,33>]
[<MailDaemon,0>, <NORRestaurantFinding,33>]
```

This is because the idea of IA is to propose all possible individual service candidates (and all possible service composition candidates that are composed from the service candidates) to the end user. In other words, it is better to propose too many service candidates to the end user rather than too few service candidates (in which the desired service candidates may not be

Result and Discussion

found). In fact, this is a tradeoff between the correctness (i.e. the desired service candidate is discovered/composed and deployed) and the performance (i.e. how well the discovered service candidates correspond to the user request).

As we considered that the correctness is more important, our design decision is to take every possibility of the candidates. However, IA ranks the service candidates in order (according to the weights as discussed in section 3.5.5). Thus, it is likely that the user could find the desired service candidate in the first members of the list. This improves the performance of IA (at least in the user point of view). As shown in Figure 122, the first desired candidate is ranked 10th in the list. In fact, the order of the list is more accurate when the IA is supplement with knowledge of NLP-tool. This will be illustrated later in this section.

A promising result is Recall (for service individual) of the list of composition candidates. The recall is equal to 100 %. This means that we have 10 relevant services retrieved (*GlobalSMS*, *VEGRestaurantFinding*, *SmsSender*, *MailDaemon*, *MessagingServer*, *PRICERestaurantFinding*, *RestaurantFinding*, *NORRestaurantFinding*, *GetRestaurantByAddress*, *GetRestaurantByCoords*) and there are only these ten services in the service repository (service repository that we used to test IA is provided in appendix C) that are relevant to the request. This indicates the highest performance possible for this scenario.

Secondly, IA processes a request by using the keyword search, ontology relation and knowledge from NLP-tool (the MRS functionality is enabled). The composition candidates are shown in Figure 123. While the MRS functionality is enabled, IA validates the composition candidates in Figure 122 with NLP-tool result. The composition candidates that do not agree with the NLP-tool result will be degraded (but not removed). Hence, the precision index and recall are the same as those of Figure 122. However, the ranking of the service candidates is more accurate. Essentially, the first desired candidate is ranked 1st in the list.

We evaluated the IA performance on the context-handling functionality (how well IA satisfies the user proposition as defined in *ContextRule.txt*). To accomplish user scenario 1, as discussed in section 5.1.1, the end user defines a service composition to be invoked when the certain user location (e.g. street@telin.nl, restaurant@telin.nl) has been changed. If the user location has been changed to street@telin.nl (the user is on the way to the restaurant), IA constructs a service composition as shown in Figure 124. It is easily to see that the composition and the input instance for service components (e.g. Paris is an input instance of *CITYTrafficReport*) are exactly the same with what is defined in *ContextRule.txt*. Thus, IA performance (with regards to preciseness of service composition) is 100 %. Clearly, this is because the user manually states the service composition and its service input parameters.

If the user location has been changed to restaurant@telin.nl (the user is at the restaurant), IA constructs a service composition as shown in Figure 125. It is easily to see that the composition and the input instance (e.g. Paris is an input instance of *WeatherReport*) for service components are exactly the same with what is defined in *ContextRule.txt*. Thus, IA performance (with regards to preciseness of service composition) is 100 %. Clearly, this is because the user manually states the service composition and its service input parameters.

Query response time for processing the request “Find vegetarian restaurant in my area and send it by sms or email” (and MRS is not utilized) is equal to 30390 ms. In other words, IA requires 30390 ms to process the request (i.e. derive and rank service composition candidates) giving that there are 141 services in the repository. Query response time for processing the request

Result and Discussion

“Find vegetarian restaurant in my area and send it by sms or email” (and MRS is utilized) is equal to 45797 ms. In other words, IA requires 45797 ms to process the request (i.e. derive and rank service composition candidates) giving that there are 141 services in the repository. Hence, IA requires 15407 ms (i.e. 45797 - 30390) more to process the request if the MRS is utilized. This brings about 50.69 % increased in the execution time if MRS is utilized. This increment is mainly caused by the standard I/O operation (i.e. The MRS of user request is stored in an XML file. IA parses and analyzes the XML file when the MRS feature is turned on). In the future work (where the MRS functionality is integrated properly (e.g. IA does not need the MRS result from external file)), we expect an improvement on the response time. This is based on an assumption that the efficiency of the grammar and efficiency of the parses are the same as before.

++List of Composition Candidates sorted by total weight++
[<SmsSender,0>, <UKLocation,50>, <VEGRestaurantFinding,8>]
[<GlobalSMS,0>, <UKLocation,50>, <VEGRestaurantFinding,8>]
[<MessagingServer,0>, <UKLocation,50>, <VEGRestaurantFinding,8>]
[<MessagingServer,0>, <VEGRestaurantFinding,33>, <UKLocationPro,14>]
[<GlobalSMS,0>, <VEGRestaurantFinding,33>, <UKLocationPro,14>]
[<SmsSender,0>, <VEGRestaurantFinding,33>, <UKLocationPro,14>]
[<MailDaemon,0>, <UKLocation,50>, <VEGRestaurantFinding,8>]
[<MailDaemon,0>, <VEGRestaurantFinding,33>, <UKLocationPro,14>]
[<MessagingServer,0>, <VEGRestaurantFinding,33>]
[<GlobalSMS,0>, <VEGRestaurantFinding,33>]
[<SmsSender,0>, <VEGRestaurantFinding,33>]
[<MailDaemon,0>, <VEGRestaurantFinding,33>]
[<GlobalSMS,0>, <UKLocation,50>, <PRICERestaurantFinding,14>]
[<MessagingServer,0>, <UKLocation,50>, <PRICERestaurantFinding,14>]
[<SmsSender,0>, <UKLocation,50>, <PRICERestaurantFinding,14>]
[<SmsSender,0>, <UKLocation,50>, <RestaurantFinding,8>]
[<GlobalSMS,0>, <UKLocation,50>, <RestaurantFinding,8>]
[<MessagingServer,0>, <UKLocation,50>, <RestaurantFinding,8>]
[<MessagingServer,0>, <UKLocation,50>, <NORRestaurantFinding,8>]
[<SmsSender,0>, <UKLocation,50>, <NORRestaurantFinding,8>]
[<GlobalSMS,0>, <UKLocation,50>, <NORRestaurantFinding,8>]
[<SmsSender,0>, <UKLocation,50>, <GetRestaurantByAddress,7>]
[<MessagingServer,0>, <UKLocation,50>, <GetRestaurantByAddress,7>]
[<GlobalSMS,0>, <UKLocation,50>, <GetRestaurantByCoords,7>]
[<SmsSender,0>, <UKLocation,50>, <GetRestaurantByCoords,7>]
[<MessagingServer,0>, <UKLocation,50>, <GetRestaurantByCoords,7>]
[<GlobalSMS,0>, <UKLocation,50>, <GetRestaurantByAddress,7>]
[<SmsSender,0>, <UKLocationPro,50>, <PRICERestaurantFinding,1>]
[<MessagingServer,0>, <UKLocationPro,50>, <PRICERestaurantFinding,1>]
[<GlobalSMS,0>, <UKLocationPro,50>, <PRICERestaurantFinding,1>]
[<GlobalSMS,0>, <UKLocationPro,50>, <GetRestaurantByAddress,5>]
[<MessagingServer,0>, <UKLocationPro,50>, <GetRestaurantByAddress,5>]
[<SmsSender,0>, <UKLocationPro,50>, <GetRestaurantByCoords,5>]
[<GlobalSMS,0>, <UKLocationPro,50>, <GetRestaurantByCoords,5>]
[<MessagingServer,0>, <UKLocationPro,50>, <GetRestaurantByCoords,5>]
[<SmsSender,0>, <UKLocationPro,50>, <GetRestaurantByAddress,5>]
[<MessagingServer,0>, <RestaurantFinding,33>, <UKLocationPro,14>]
[<GlobalSMS,0>, <RestaurantFinding,33>, <UKLocationPro,14>]
[<SmsSender,0>, <RestaurantFinding,33>, <UKLocationPro,14>]
[<SmsSender,0>, <NORRestaurantFinding,33>, <UKLocationPro,14>]
[<MessagingServer,0>, <NORRestaurantFinding,33>, <UKLocationPro,14>]
[<GlobalSMS,0>, <NORRestaurantFinding,33>, <UKLocationPro,14>]
[<MailDaemon,0>, <UKLocation,50>, <PRICERestaurantFinding,14>]
[<GlobalSMS,0>, <PRICERestaurantFinding,50>]

Result and Discussion

```
[<MessagingServer,0>, <PRICERestaurantFinding,50>]
[<SmsSender,0>, <PRICERestaurantFinding,50>]
[<MailDaemon,0>, <UKLocation,50>, <RestaurantFinding,8>]
[<MailDaemon,0>, <UKLocation,50>, <NORRestaurantFinding,8>]
[<MailDaemon,0>, <UKLocation,50>, <GetRestaurantByCoords,7>]
[<MailDaemon,0>, <UKLocation,50>, <GetRestaurantByAddress,7>]
[<MailDaemon,0>, <UKLocationPro,50>, <PRICERestaurantFinding,1>]
[<MailDaemon,0>, <UKLocationPro,50>, <GetRestaurantByCoords,5>]
[<MailDaemon,0>, <UKLocationPro,50>, <GetRestaurantByAddress,5>]
[<MailDaemon,0>, <RestaurantFinding,33>, <UKLocationPro,14>]
[<MailDaemon,0>, <NORRestaurantFinding,33>, <UKLocationPro,14>]
[<MessagingServer,0>, <RestaurantFinding,33>]
[<GlobalSMS,0>, <RestaurantFinding,33>]
[<SmsSender,0>, <RestaurantFinding,33>]
[<SmsSender,0>, <NORRestaurantFinding,33>]
[<MessagingServer,0>, <NORRestaurantFinding,33>]
[<GlobalSMS,0>, <NORRestaurantFinding,33>]
[<VEGRestaurantFinding,0>, <UKLocationPro,14>]
[<MailDaemon,0>, <PRICERestaurantFinding,50>]
[<SmsSender,0>, <UKLocationPro,50>]
[<GlobalSMS,0>, <UKLocation,50>]
[<MessagingServer,0>, <UKLocationPro,50>]
[<MessagingServer,0>, <UKLocation,50>]
[<GlobalSMS,0>, <UKLocationPro,50>]
[<SmsSender,0>, <UKLocation,50>]
[<UKLocation,0>, <VEGRestaurantFinding,8>]
[<MailDaemon,0>, <RestaurantFinding,33>]
[<MailDaemon,0>, <NORRestaurantFinding,33>]
[<GlobalSMS,0>, <MailLocate,5>]
[<SmsSender,0>, <MailLocate,5>]
[<VEGRestaurantFinding,0>]
[<MailDaemon,0>, <UKLocation,50>]
[<MailDaemon,0>, <UKLocationPro,50>]
[<MailDaemon,0>, <MailLocate,20>]
[<MessagingServer,0>, <MailLocate,20>]
[<GlobalSMS,0>]
[<SmsSender,0>]
[<MailDaemon,0>]
[<RestaurantFinding,0>, <UKLocationPro,14>]
[<UKLocation,0>, <PRICERestaurantFinding,14>]
[<NORRestaurantFinding,0>, <UKLocationPro,14>]
[<UKLocation,0>, <RestaurantFinding,8>]
[<UKLocation,0>, <NORRestaurantFinding,8>]
[<UKLocation,0>, <GetRestaurantByCoords,7>]
[<UKLocation,0>, <GetRestaurantByAddress,7>]
[<UKLocationPro,0>, <PRICERestaurantFinding,1>]
[<UKLocationPro,0>, <GetRestaurantByCoords,5>]
[<UKLocationPro,0>, <GetRestaurantByAddress,5>]
[<MessagingServer,0>]
[<RestaurantFinding,0>]
[<PRICERestaurantFinding,0>]
[<NORRestaurantFinding,0>]
[<MailLocate,0>]
[<SkyScanner,0>]
[<UKLocationPro,0>]
[<UKLocation,0>]
end ++List of Composition Candidates sorted by total weight++
```

Figure 122: List of Composition Candidates for scenario 1 (MRS disabled)

Result and Discussion

```
++List of Composition Candidates sorted by nlp++
[<MessagingServer,0>, <VEGRestaurantFinding,33>]
[<GlobalSMS,0>, <VEGRestaurantFinding,33>]
[<SmsSender,0>, <VEGRestaurantFinding,33>]
[<MailDaemon,0>, <VEGRestaurantFinding,33>]
[<GlobalSMS,0>, <PRICERestaurantFinding,50>]
[<MessagingServer,0>, <PRICERestaurantFinding,50>]
[<SmsSender,0>, <PRICERestaurantFinding,50>]
[<MessagingServer,0>, <RestaurantFinding,33>]
[<GlobalSMS,0>, <RestaurantFinding,33>]
[<SmsSender,0>, <RestaurantFinding,33>]
[<SmsSender,0>, <NORRestaurantFinding,33>]
[<MessagingServer,0>, <NORRestaurantFinding,33>]
[<GlobalSMS,0>, <NORRestaurantFinding,33>]
[<MailDaemon,0>, <PRICERestaurantFinding,50>]
[<SmsSender,0>, <UKLocationPro,50>]
[<GlobalSMS,0>, <UKLocation,50>]
[<MessagingServer,0>, <UKLocationPro,50>]
[<MessagingServer,0>, <UKLocation,50>]
[<GlobalSMS,0>, <UKLocationPro,50>]
[<SmsSender,0>, <UKLocation,50>]
[<UKLocation,0>, <VEGRestaurantFinding,8>]
[<MailDaemon,0>, <RestaurantFinding,33>]
[<MailDaemon,0>, <NORRestaurantFinding,33>]
[<GlobalSMS,0>, <MailLocate,5>]
[<SmsSender,0>, <MailLocate,5>]
[<MailDaemon,0>, <UKLocation,50>]
[<MailDaemon,0>, <UKLocationPro,50>]
[<MailDaemon,0>, <MailLocate,20>]
[<MessagingServer,0>, <MailLocate,20>]
[<UKLocation,0>, <PRICERestaurantFinding,14>]
[<UKLocation,0>, <RestaurantFinding,8>]
[<UKLocation,0>, <NORRestaurantFinding,8>]
[<UKLocation,0>, <GetRestaurantByCoords,7>]
[<UKLocation,0>, <GetRestaurantByAddress,7>]
[<UKLocationPro,0>, <PRICERestaurantFinding,1>]
[<UKLocationPro,0>, <GetRestaurantByCoords,5>]
[<UKLocationPro,0>, <GetRestaurantByAddress,5>]
[<SmsSender,0>, <UKLocation,50>, <VEGRestaurantFinding,8>]
[<GlobalSMS,0>, <UKLocation,50>, <VEGRestaurantFinding,8>]
[<MessagingServer,0>, <UKLocation,50>, <VEGRestaurantFinding,8>]
[<MessagingServer,0>, <VEGRestaurantFinding,33>, <UKLocationPro,14>]
[<GlobalSMS,0>, <VEGRestaurantFinding,33>, <UKLocationPro,14>]
[<SmsSender,0>, <VEGRestaurantFinding,33>, <UKLocationPro,14>]
[<MailDaemon,0>, <UKLocation,50>, <VEGRestaurantFinding,8>]
[<MailDaemon,0>, <VEGRestaurantFinding,33>, <UKLocationPro,14>]
[<GlobalSMS,0>, <UKLocation,50>, <PRICERestaurantFinding,14>]
[<MessagingServer,0>, <UKLocation,50>, <PRICERestaurantFinding,14>]
[<SmsSender,0>, <UKLocation,50>, <PRICERestaurantFinding,14>]
[<SmsSender,0>, <UKLocation,50>, <RestaurantFinding,8>]
[<GlobalSMS,0>, <UKLocation,50>, <RestaurantFinding,8>]
[<MessagingServer,0>, <UKLocation,50>, <RestaurantFinding,8>]
[<MessagingServer,0>, <UKLocation,50>, <NORRestaurantFinding,8>]
[<SmsSender,0>, <UKLocation,50>, <NORRestaurantFinding,8>]
[<GlobalSMS,0>, <UKLocation,50>, <NORRestaurantFinding,8>]
[<SmsSender,0>, <UKLocation,50>, <GetRestaurantByAddress,7>]
[<MessagingServer,0>, <UKLocation,50>, <GetRestaurantByAddress,7>]
[<GlobalSMS,0>, <UKLocation,50>, <GetRestaurantByCoords,7>]
[<SmsSender,0>, <UKLocation,50>, <GetRestaurantByCoords,7>]
[<MessagingServer,0>, <UKLocation,50>, <GetRestaurantByCoords,7>]
```


Result and Discussion

```
[<GlobalSMS,0>, <UKLocation,50>, <GetRestaurantByAddress,7>]
[<SmsSender,0>, <UKLocationPro,50>, <PRICERestaurantFinding,1>]
[<MessagingServer,0>, <UKLocationPro,50>, <PRICERestaurantFinding,1>]
[<GlobalSMS,0>, <UKLocationPro,50>, <PRICERestaurantFinding,1>]
[<GlobalSMS,0>, <UKLocationPro,50>, <GetRestaurantByAddress,5>]
[<MessagingServer,0>, <UKLocationPro,50>, <GetRestaurantByAddress,5>]
[<SmsSender,0>, <UKLocationPro,50>, <GetRestaurantByCoords,5>]
[<GlobalSMS,0>, <UKLocationPro,50>, <GetRestaurantByCoords,5>]
[<MessagingServer,0>, <UKLocationPro,50>, <GetRestaurantByCoords,5>]
[<SmsSender,0>, <UKLocationPro,50>, <GetRestaurantByAddress,5>]
[<MessagingServer,0>, <RestaurantFinding,33>, <UKLocationPro,14>]
[<GlobalSMS,0>, <RestaurantFinding,33>, <UKLocationPro,14>]
[<SmsSender,0>, <RestaurantFinding,33>, <UKLocationPro,14>]
[<SmsSender,0>, <NORRestaurantFinding,33>, <UKLocationPro,14>]
[<MessagingServer,0>, <NORRestaurantFinding,33>, <UKLocationPro,14>]
[<GlobalSMS,0>, <NORRestaurantFinding,33>, <UKLocationPro,14>]
[<MailDaemon,0>, <UKLocation,50>, <PRICERestaurantFinding,14>]
[<MailDaemon,0>, <UKLocation,50>, <RestaurantFinding,8>]
[<MailDaemon,0>, <UKLocation,50>, <NORRestaurantFinding,8>]
[<MailDaemon,0>, <UKLocation,50>, <GetRestaurantByCoords,7>]
[<MailDaemon,0>, <UKLocation,50>, <GetRestaurantByAddress,7>]
[<MailDaemon,0>, <UKLocationPro,50>, <PRICERestaurantFinding,1>]
[<MailDaemon,0>, <UKLocationPro,50>, <GetRestaurantByCoords,5>]
[<MailDaemon,0>, <UKLocationPro,50>, <GetRestaurantByAddress,5>]
[<MailDaemon,0>, <RestaurantFinding,33>, <UKLocationPro,14>]
[<MailDaemon,0>, <NORRestaurantFinding,33>, <UKLocationPro,14>]
[<VEGRestaurantFinding,0>, <UKLocationPro,14>]
[<VEGRestaurantFinding,0>]
[<GlobalSMS,0>]
[<SmsSender,0>]
[<MailDaemon,0>]
[<RestaurantFinding,0>, <UKLocationPro,14>]
[<NORRestaurantFinding,0>, <UKLocationPro,14>]
[<MessagingServer,0>]
[<RestaurantFinding,0>]
[<PRICERestaurantFinding,0>]
[<NORRestaurantFinding,0>]
[<MailLocate,0>]
[<SkyScanner,0>]
[<UKLocationPro,0>]
[<UKLocation,0>]
```

end ++List of Composition Candidates sorted by nlp++

Figure 123: List of Composition Candidates for scenario 1 (MRS enabled)

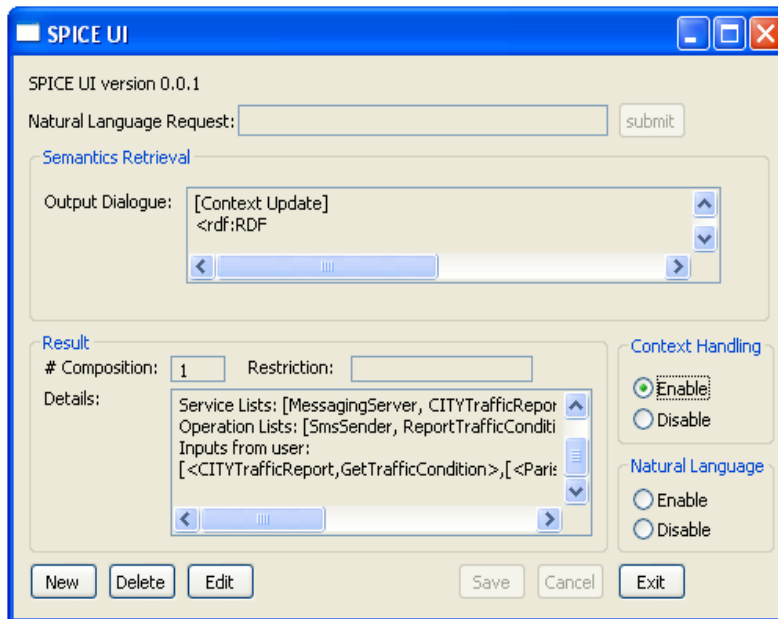


Figure 124: IA constructs a service composition when the user location has been changed to street@telin.nl

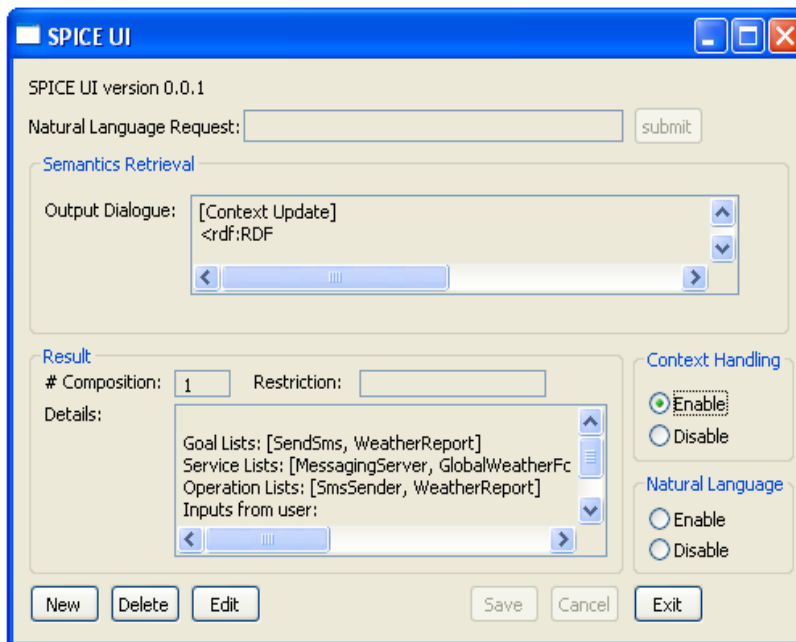


Figure 125: IA constructs a service composition when the user location has been changed to restaurant@telin.nl

5.4.3 Discussion on performance: Scenario 2

As discussed in section 5.1.2, a user submits a request “Find weather report in Paris, translated to English and send it by email”. IA processes a request in two alternative ways. Firstly, IA processes a request by using only the keyword search and ontology relation (the MRS functionality is disable). The composition candidates are shown in Figure 126.

Result and Discussion

Precision index (for service individual) of the list of composition candidates is equal to 50.00 %. This means that out of 12 services retrieved (*GoogleTranslate*, *WeatherForecastPro*, *GlovalWeatherForecast*, *MailDaemon*, *GlobalSMS*, *WeatherForecast*, *WeatherByCity*, *MessagingServer*, *MailLocate*, *SkyScanner*, *UKLocationPro*, *UKLocation*), there are four services unrelated to the request. The four services are *UKLocationPro*, *UKLocation*, *MailLocate* and *SkyScanner*. *UKLocation* and *UKLocationPro* are two services related to find a location of the user in UK. *MailLocate* is a service to find an email address of the particular user. *GlobalSMS* and *MessagingServer* are two services related to deliver SMS to the user. *SkyScanner* is a service to search for a flight.

Thus, this gives a precision index (for service individual) of $(12 - 6)/12$ which is 50.00 %. This indicates an average performance. Precision index (for service composition) of the list of composition candidates is equal to 6.4 %. This means that out of 77 service compositions (Figure 126), there are only three compositions relevant to the request. The three compositions are:

```
[<MailDaemon,0>, <GoogleTranslate,50>, <WeatherForecastPro,16>]  
[<MailDaemon,0>, <GoogleTranslate,50>, <GlobalWeatherForecast,16>]  
[<MailDaemon,0>, <GoogleTranslate,50>, <WeatherByCity,16>]
```

This is because the idea of IA is to propose all possible individual service candidates (and all possible service composition candidates that are composed from the service candidates) to the end user. In other words, it is better to propose too many service candidates to the end user rather than too few service candidates (in which the desired service candidates may not be found). In fact, this is a tradeoff between the correctness (i.e. the desired service candidate is discovered/composed and deployed) and the performance (i.e. how well the discovered service candidates correspond to the user request). As we considered that the correctness is more important, our design decision is to take every possibility of the candidates. However, IA ranks the service candidates in order (according to the weights as discussed in section 3.5.5). Thus, it is likely that the user could find the desired service candidate in the first members of the list. This improves the performance of IA (at least in the user point of view). As shown in Figure 126, the first desired candidate is ranked 7th in the list. In fact, the order of the list is more accurate when the IA is supplement with knowledge of NLP-tool. This will be illustrated later in this section.

A promising result is Recall (for service individual) of the list of composition candidates. The recall is equal to 100 %. This means that we have 5 relevant services retrieved (*GoogleTranslate*, *WeatherForecastPro*, *GlovalWeatherForecast*, *MailDaemon*, *WeatherForecast*, *WeatherByCity*) and there are only these five services in the service repository that are relevant to the request. This indicates the highest performance possible for this scenario.

Secondly, IA processes a request by using the keyword search, ontology relation and knowledge from NLP-tool (the MRS functionality is enabled). The composition candidates are shown in Figure 127. While the MRS functionality is enabled, IA validates the composition candidates in Figure 126 with NLP-tool result. The composition candidates that do not agree with the NLP-tool result will be degraded (but not removed). Hence, the precision index and recall are the same as those of Figure 126. However, the ranking of the service candidates is more accurate. Essentially, the first desired candidate is ranked 1st in the list.

Query response time for processing the request “Find weather report in Paris, translated to English and send it by email” (and MRS is not utilized) is equal to 13657 ms. In other words, IA requires 13657 ms to process the request (i.e. derive and rank service composition candi-

Result and Discussion

dates) giving that there are 141 services in the repository. Query response time for processing the request “Find weather report in Paris, translated to English and send it by email” (and MRS is utilized) is equal to 16719 ms. In other words, IA requires 16719 ms to process the request (i.e. derive and rank service composition candidates) giving that there are 141 services in the repository. Hence, IA requires 3062 ms (i.e. 16719 - 13657) more to process the request if the MRS is utilized. This brings about 22.42 % increased in the execution time if MRS is utilized. This increment is mainly caused by the standard I/O operation (i.e. The MRS of user request is stored in an XML file. IA parses and analyzes the XML file when the MRS feature is turned on). In the future work (where the MRS functionality is integrated properly (e.g. IA does not need the MRS result from external file)), we expect an improvement on the response time. This is based on an assumption that the efficiency of the grammar and efficiency of the parses are the same as before.

++List of Composition Candidates sorted by total weight++
[<MailDaemon,0>, <GoogleTranslate,50>, <UKLocation,14>, <GlobalWeatherForecast,14>]
[<MailDaemon,0>, <GoogleTranslate,50>, <UKLocation,14>, <WeatherForecastPro,14>]
[<GlobalSMS,0>, <GoogleTranslate,50>, <UKLocation,14>, <WeatherForecastPro,14>]
[<MessagingServer,0>, <GoogleTranslate,50>, <UKLocation,14>, <GlobalWeatherForecast,14>]
[<MessagingServer,0>, <GoogleTranslate,50>, <UKLocation,14>, <WeatherForecastPro,14>]
[<GlobalSMS,0>, <GoogleTranslate,50>, <UKLocation,14>, <GlobalWeatherForecast,14>]
[<MailDaemon,0>, <GoogleTranslate,50>, <WeatherForecastPro,16>]
[<MailDaemon,0>, <GoogleTranslate,50>, <GlobalWeatherForecast,16>]
[<MailDaemon,0>, <GoogleTranslate,50>, <UKLocation,14>, <WeatherByCity,14>]
[<GlobalSMS,0>, <GoogleTranslate,50>, <GlobalWeatherForecast,16>]
[<GlobalSMS,0>, <GoogleTranslate,50>, <WeatherForecastPro,16>]
[<MessagingServer,0>, <GoogleTranslate,50>, <GlobalWeatherForecast,16>]
[<MessagingServer,0>, <GoogleTranslate,50>, <WeatherForecastPro,16>]
[<MessagingServer,0>, <GoogleTranslate,50>, <UKLocation,14>, <WeatherByCity,14>]
[<GlobalSMS,0>, <GoogleTranslate,50>, <UKLocation,14>, <WeatherByCity,14>]
[<MailDaemon,0>, <GoogleTranslate,50>, <WeatherByCity,16>]
[<GlobalSMS,0>, <GoogleTranslate,50>, <WeatherByCity,16>]
[<MessagingServer,0>, <GoogleTranslate,50>, <WeatherByCity,16>]
[<MailDaemon,0>, <GoogleTranslate,50>, <UKLocationPro,14>]
[<MailDaemon,0>, <GoogleTranslate,50>, <UKLocation,14>]
[<MailDaemon,0>, <UKLocation,50>, <WeatherForecastPro,14>]
[<MailDaemon,0>, <UKLocation,50>, <GlobalWeatherForecast,14>]
[<GlobalSMS,0>, <GoogleTranslate,50>, <UKLocation,14>]
[<MessagingServer,0>, <GoogleTranslate,50>, <UKLocationPro,14>]
[<MessagingServer,0>, <GoogleTranslate,50>, <UKLocation,14>]
[<GlobalSMS,0>, <GoogleTranslate,50>, <UKLocationPro,14>]
[<MailDaemon,0>, <GoogleTranslate,50>]
[<GoogleTranslate,0>, <UKLocation,14>, <GlobalWeatherForecast,14>]
[<GoogleTranslate,0>, <UKLocation,14>, <WeatherForecastPro,14>]
[<MessagingServer,0>, <UKLocation,50>, <WeatherForecastPro,14>]
[<GlobalSMS,0>, <UKLocation,50>, <GlobalWeatherForecast,14>]
[<GlobalSMS,0>, <UKLocation,50>, <WeatherForecastPro,14>]
[<MessagingServer,0>, <UKLocation,50>, <GlobalWeatherForecast,14>]
[<MailDaemon,0>, <GlobalWeatherForecast,50>]
[<MailDaemon,0>, <WeatherForecastPro,50>]
[<MailDaemon,0>, <UKLocation,50>, <WeatherByCity,14>]
[<GlobalSMS,0>, <GoogleTranslate,50>]
[<MessagingServer,0>, <GoogleTranslate,50>]
[<GoogleTranslate,0>, <WeatherForecastPro,16>]
[<GoogleTranslate,0>, <GlobalWeatherForecast,16>]
[<GoogleTranslate,0>, <UKLocation,14>, <WeatherByCity,14>]
[<GlobalSMS,0>, <WeatherForecastPro,50>]
[<MessagingServer,0>, <GlobalWeatherForecast,50>]

Result and Discussion

```
[<MessagingServer,0>, <WeatherForecastPro,50>]
[<GlobalSMS,0>, <GlobalWeatherForecast,50>]
[<GlobalSMS,0>, <UKLocation,50>, <WeatherByCity,14>]
[<MessagingServer,0>, <UKLocation,50>, <WeatherByCity,14>]
[<MailDaemon,0>, <WeatherByCity,50>]
[<GoogleTranslate,0>, <WeatherByCity,16>]
[<GlobalSMS,0>, <WeatherByCity,50>]
[<MessagingServer,0>, <WeatherByCity,50>]
[<MailDaemon,0>, <UKLocation,50>]
[<MailDaemon,0>, <UKLocationPro,50>]
[<MailDaemon,0>, <MailLocate,20>]
[<GoogleTranslate,0>, <UKLocationPro,14>]
[<GoogleTranslate,0>, <UKLocation,14>]
[<GlobalSMS,0>, <UKLocationPro,50>]
[<GlobalSMS,0>, <UKLocation,50>]
[<MessagingServer,0>, <UKLocationPro,50>]
[<MessagingServer,0>, <UKLocation,50>]
[<UKLocation,0>, <GlobalWeatherForecast,14>]
[<UKLocation,0>, <WeatherForecastPro,14>]
[<MessagingServer,0>, <MailLocate,20>]
[<GoogleTranslate,0>]
[<GlobalSMS,0>, <MailLocate,5>]
[<WeatherForecastPro,0>]
[<GlobalWeatherForecast,0>]
[<UKLocation,0>, <WeatherByCity,14>]
[<MailDaemon,0>]
[<GlobalSMS,0>]
[<WeatherForecast,0>]
[<WeatherByCity,0>]
[<MessagingServer,0>]
[<MailLocate,0>]
[<SkyScanner,0>]
[<UKLocationPro,0>]
[<UKLocation,0>]
end ++List of Composition Candidates sorted by total weight++
```

Figure 126: List of Composition Candidates for scenario 1 (MRS disabled)

```
++List of Composition Candidates sorted by nlp++
[<MailDaemon,0>, <GoogleTranslate,50>, <WeatherForecastPro,16>]
[<MailDaemon,0>, <GoogleTranslate,50>, <GlobalWeatherForecast,16>]
[<GlobalSMS,0>, <GoogleTranslate,50>, <GlobalWeatherForecast,16>]
[<GlobalSMS,0>, <GoogleTranslate,50>, <WeatherForecastPro,16>]
[<MessagingServer,0>, <GoogleTranslate,50>, <GlobalWeatherForecast,16>]
[<MessagingServer,0>, <GoogleTranslate,50>, <WeatherForecastPro,16>]
[<MailDaemon,0>, <GoogleTranslate,50>, <WeatherByCity,16>]
[<GlobalSMS,0>, <GoogleTranslate,50>, <WeatherByCity,16>]
[<MessagingServer,0>, <GoogleTranslate,50>, <WeatherByCity,16>]
[<MailDaemon,0>, <UKLocation,50>, <WeatherForecastPro,14>]
[<MailDaemon,0>, <UKLocation,50>, <GlobalWeatherForecast,14>]
[<MessagingServer,0>, <UKLocation,50>, <WeatherForecastPro,14>]
[<GlobalSMS,0>, <UKLocation,50>, <GlobalWeatherForecast,14>]
[<GlobalSMS,0>, <UKLocation,50>, <WeatherForecastPro,14>]
[<MessagingServer,0>, <UKLocation,50>, <GlobalWeatherForecast,14>]
[<MailDaemon,0>, <UKLocation,50>, <WeatherByCity,14>]
[<GlobalSMS,0>, <UKLocation,50>, <WeatherByCity,14>]
[<MessagingServer,0>, <UKLocation,50>, <WeatherByCity,14>]
[<MailDaemon,0>, <GoogleTranslate,50>, <UKLocation,14>, <GlobalWeatherForecast,14>]
[<MailDaemon,0>, <GoogleTranslate,50>, <UKLocation,14>, <WeatherForecastPro,14>]
[<GlobalSMS,0>, <GoogleTranslate,50>, <UKLocation,14>, <WeatherForecastPro,14>]
[<MessagingServer,0>, <GoogleTranslate,50>, <UKLocation,14>, <GlobalWeatherForecast,14>]
```

Result and Discussion

```
[<MessagingServer,0>, <GoogleTranslate,50>, <UKLocation,14>, <WeatherForecastPro,14>]
[<GlobalSMS,0>, <GoogleTranslate,50>, <UKLocation,14>, <GlobalWeatherForecast,14>]
[<MailDaemon,0>, <GoogleTranslate,50>, <UKLocation,14>, <WeatherByCity,14>]
[<MessagingServer,0>, <GoogleTranslate,50>, <UKLocation,14>, <WeatherByCity,14>]
[<GlobalSMS,0>, <GoogleTranslate,50>, <UKLocation,14>, <WeatherByCity,14>]
[<MailDaemon,0>, <GoogleTranslate,50>, <UKLocationPro,14>]
[<MailDaemon,0>, <GoogleTranslate,50>, <UKLocation,14>]
[<GlobalSMS,0>, <GoogleTranslate,50>, <UKLocation,14>]
[<MessagingServer,0>, <GoogleTranslate,50>, <UKLocationPro,14>]
[<MessagingServer,0>, <GoogleTranslate,50>, <UKLocation,14>]
[<GlobalSMS,0>, <GoogleTranslate,50>, <UKLocationPro,14>]
[<MailDaemon,0>, <GoogleTranslate,50>]
[<GoogleTranslate,0>, <UKLocation,14>, <GlobalWeatherForecast,14>]
[<GoogleTranslate,0>, <UKLocation,14>, <WeatherForecastPro,14>]
[<MailDaemon,0>, <GlobalWeatherForecast,50>]
[<MailDaemon,0>, <WeatherForecastPro,50>]
[<GlobalSMS,0>, <GoogleTranslate,50>]
[<MessagingServer,0>, <GoogleTranslate,50>]
[<GoogleTranslate,0>, <WeatherForecastPro,16>]
[<GoogleTranslate,0>, <GlobalWeatherForecast,16>]
[<GoogleTranslate,0>, <UKLocation,14>, <WeatherByCity,14>]
[<GlobalSMS,0>, <WeatherForecastPro,50>]
[<MessagingServer,0>, <GlobalWeatherForecast,50>]
[<MessagingServer,0>, <WeatherForecastPro,50>]
[<GlobalSMS,0>, <GlobalWeatherForecast,50>]
[<MailDaemon,0>, <WeatherByCity,50>]
[<GoogleTranslate,0>, <WeatherByCity,16>]
[<GlobalSMS,0>, <WeatherByCity,50>]
[<MessagingServer,0>, <WeatherByCity,50>]
[<MailDaemon,0>, <UKLocation,50>]
[<MailDaemon,0>, <UKLocationPro,50>]
[<MailDaemon,0>, <MailLocate,20>]
[<GoogleTranslate,0>, <UKLocationPro,14>]
[<GoogleTranslate,0>, <UKLocation,14>]
[<GlobalSMS,0>, <UKLocationPro,50>]
[<GlobalSMS,0>, <UKLocation,50>]
[<MessagingServer,0>, <UKLocationPro,50>]
[<MessagingServer,0>, <UKLocation,50>]
[<UKLocation,0>, <GlobalWeatherForecast,14>]
[<UKLocation,0>, <WeatherForecastPro,14>]
[<MessagingServer,0>, <MailLocate,20>]
[<GoogleTranslate,0>]
[<GlobalSMS,0>, <MailLocate,5>]
[<WeatherForecastPro,0>]
[<GlobalWeatherForecast,0>]
[<UKLocation,0>, <WeatherByCity,14>]
[<MailDaemon,0>]
[<GlobalSMS,0>]
[<WeatherForecast,0>]
[<WeatherByCity,0>]
[<MessagingServer,0>]
[<MailLocate,0>]
[<SkyScanner,0>]
[<UKLocationPro,0>]
[<UKLocation,0>]
end ++List of Composition Candidates sorted by nlp++
```

Figure 127: List of Composition Candidates for scenario 1 (MRS enabled)

As discussed in section 5.1.2, a user also submits a request “Search a flight from London to Paris on 10-06-2008 and book it”. IA processes a request in two alternative ways. Firstly, IA processes a request by using only the keyword search and ontology relation (the MRS functionality is disable). The composition candidates are shown in Figure 128.

Precision index (for service individual) of the list of composition candidates is equal to 40.00 %. This means that out of 5 services retrieved (*FlightBooking*, *SkyScanner*, *RestaurantBooking*, *UKLocation*, *UKLocationPro*), there are three services unrelated to the request. The four services are *RestaurantBooking*, *UKLocation* and *UKLocationPro*. *UKLocation* and *UKLocationPro* are two services related to find a location of the user in UK. *RestaurantBooking* is a service that is used to book for a restaurant. Thus, this gives a precision index (for service individual) of $(5 - 3)/5$ which is 40.00 %. This indicates a lower performance than the traditional keyword-based ways of service discovery (for this particular scenario) [50].

The reason is that the services in the service repository (shown in appendix C) are not distributed enough. There are only two services associated with the word “booking” and this makes the word “booking” has very high weight (the low number of service composition candidates (i.e. 8 service composition candidates have been retrieved) also emphasize this fact). In other words, the service goal *RestaurantBooking* is selected because of the keyword, “booking”, is in the user request (i.e. the weight of the “booking” is larger than a threshold (a half) of the weight of “restaurant” + the weight “booking”). If the services in the service repository were distributed enough, one keyword would not be able to justify the selection of service component. Hence, the performance of IA will be better. This indicates a disadvantage of service discovery based on keyword-matching (i.e. irrelevant services get selected).

Precision index (for service composition) of the list of composition candidates is equal to 12.5 %. This means that out of 8 service compositions (Figure 129), there are only one compositions relevant to the request. The composition is:

[<FlightBooking,0>, <SkyScanner,100>]

This is because the idea of IA is to propose all possible individual service candidates (and all possible service composition candidates that are composed from the service candidates) to the end user. In other words, it is better to propose too many service candidates to the end user rather than too few service candidates (in which the desired service candidates may not be found). In fact, this is a tradeoff between the correctness (i.e. the desired service candidate is discovered/composed and deployed) and the performance (i.e. how well the discovered service candidates correspond to the user request). As we considered that the correctness is more important, our design decision is to take every possibility of the candidates.

However, IA ranks the service candidates in order (according to the weights as discussed in section 3.5.5). Thus, it is likely that the user could find the desired service candidate in the first members of the list. This improves the performance of IA (at least in the user point of view). As shown in Figure 129, the first desired candidate is ranked 1st in the list.

In fact, the order of the list is the same as when the IA is supplement with knowledge of NLP-tool. This is because the number of candidates is small. In other words, the maximum number of service components in service composition candidates has an influence on the ranking of the list of candidates. Normally, the higher number (of service components) gets higher rank in the list. This is because the total weight is calculated from the weight of service goal and the con-

Result and Discussion

nection weight of service components. Therefore, the service candidate that has many service components has higher chance to get higher rank than the service candidate that is composed from a few service components. However, in this scenario, the service candidate is, fortunately, composed of at most two service components (this agrees with the MRS result). Therefore, the ranking of service candidates are the same for the two ways of running IA (with MRS and without MRS). This will be illustrated later in this section.

A promising result is Recall (for service individual) of the list of composition candidates. The recall is equal to 100 %. This means that we have 2 relevant services retrieved (*FlightBooking*, *SkyScanner*) and there are only these two services in the service repository that are relevant to the request. This indicates the highest performance possible for this scenario.

Secondly, IA processes a request by using the keyword search, ontology relation and knowledge from NLP-tool (the MRS functionality is enabled). The composition candidates are shown in Figure 129. While the MRS functionality is enabled, IA validates the composition candidates in Figure 128 with NLP-tool result. The composition candidates that do not agree with the NLP-tool result will be degraded (but not removed). Hence, the precision index and recall are the same as those of Figure 128. Moreover, the ranking of the service candidates is also the same, for this case. Essentially, the first desired candidate is ranked 1st in the list.

Query response time for processing the request “Search a flight from London to Paris on 10-06-2008 and book it” (and MRS is not utilized) is equal to 1969 ms. In other words, IA requires 1969 ms to process the request (i.e. derive and rank service composition candidates) giving that there are 141 services in the repository. Query response time for processing the request “Search a flight from London to Paris on 10-06-2008 and book it” (and MRS is utilized) is equal to 2563 ms. In other words, IA requires 2563 ms to process the request (i.e. derive and rank service composition candidates) giving that there are 141 services in the repository. Hence, IA requires 594 ms (i.e. 2563 - 1969) more to process the request if the MRS is utilized. This brings about 30.16 % increased in the execution time if MRS is utilized. This increment is mainly caused by the standard I/O operation (i.e. The MRS of user request is stored in an XML file. IA parses and analyzes the XML file when the MRS feature is turned on). In the future work (where the MRS functionality is integrated properly (e.g. IA does not need the MRS result from external file)), we expect an improvement on the response time. This is based on an assumption that the efficiency of the grammar and efficiency of the parses are the same as before.

However, the query response time getting from the request is comparable to the query response time of [50] (even [50] is a query response time of service discovery (i.e. not include service composition)). This indicates a promising result when there are only few services in the repository matched with the user request.

```
++List of Composition Candidates sorted by total weight++
[<FlightBooking,0>, <SkyScanner,100>]
[<RestaurantBooking,0>, <UKLocation,14>]
[<RestaurantBooking,0>, <UKLocationPro,14>]
[<FlightBooking,0>]
[<SkyScanner,0>]
[<RestaurantBooking,0>]
[<UKLocation,0>]
[<UKLocationPro,0>]
end ++List of Composition Candidates sorted by total weight++
```

Figure 128: List of Composition Candidates for scenario 2-2 (MRS disabled)


```
++List of Composition Candidates sorted by nlp++
[<FlightBooking,0>, <SkyScanner,100>]
[<RestaurantBooking,0>, <UKLocation,14>]
[<RestaurantBooking,0>, <UKLocationPro,14>]
[<FlightBooking,0>]
[<SkyScanner,0>]
[<RestaurantBooking,0>]
[<UKLocation,0>]
[<UKLocationPro,0>]
end ++List of Composition Candidates sorted by nlp++
```

Figure 129: List of Composition Candidates for scenario 2-2 (MRS enabled)

5.4.4 Discussion on performance: Scenario 3

As discussed in section 5.1.3, a user submits a request “Find an address of theater in my area and send it to my email”. IA processes a request in two alternative ways. Firstly, IA processes a request by using only the keyword search and ontology relation (the MRS functionality is disable). The composition candidates are shown in Figure 130.

Precision index (for service individual) of the list of composition candidates is equal to 53.84 %. This means that out of 13 services retrieved (*TheaterSearch*, *CITYTheaterSearch*, *PRICTheaterSearch*, *STATTheaterSearch*, *MailDaemon*, *MessagingServer*, *GlobalSMS*, *GetRestaurantByAddress*, *MailLocate*, *URLToIPAddress*, *SkyScanner*, *UKLocation*, *UKLocationPro*), there are six services unrelated to the request. The six services are *UKLocation*, *UKLocationPro*, *GetRestaurantByAddress*, *MailLocate*, *URLToIPAddress* and *SkyScanner*. *UKLocation* and *UKLocationPro* are two services related to find a location of the user in UK. *GetRestaurantByAddress* is a service that is used to find restaurant information giving an address as an input. *MailLocate* is a service to find an email address of the particular user. *URLToIPAddress* is a service to find an IP address from the URL. *SkyScanner* is a service to search for a flight. Thus, this gives a precision index (for service individual) of $(13 - 6)/13$ which is 53.84 %. This indicates an average performance. Precision index (for service composition) of the list of composition candidates is equal to 9.3 %. This means that out of 128 service compositions (Figure 130), there are only twelve compositions relevant to the request. The twelve compositions are:

```
[<MailDaemon,0>, <TheaterSearch,50>]
[<MailDaemon,0>, <CITYTheaterSearch,50>]
[<MailDaemon,0>, <PRICTheaterSearch,50>]
[<GlobalSMS,0>, <TheaterSearch,50>]
[<MailDaemon,0>, <STATTheaterSearch,50>]
[<MessagingServer,0>, <TheaterSearch,50>]
[<GlobalSMS,0>, <CITYTheaterSearch,50>]
[<MessagingServer,0>, <CITYTheaterSearch,50>]
[<GlobalSMS,0>, <PRICTheaterSearch,50>]
[<MessagingServer,0>, <PRICTheaterSearch,50>]
[<GlobalSMS,0>, <STATTheaterSearch,50>]
[<MessagingServer,0>, <STATTheaterSearch,50>]
```

This is because the idea of IA is to propose all possible individual service candidates (and all possible service composition candidates that are composed from the service candidates) to the end user. In other words, it is better to propose too many service candidates to the end user rather than too few service candidates (in which the desired service candidates may not be found). In fact, this is a tradeoff between the correctness (i.e. the desired service candidate is

Result and Discussion

discovered/composed and deployed) and the performance (i.e. how well the discovered service candidates correspond to the user request). As we considered that the correctness is more important, our design decision is to take every possibility of the candidates. However, IA ranks the service candidates in order (according to the weights as discussed in section 3.5.5). Thus, it is likely that the user could find the desired service candidate in the first members of the list. This improves the performance of IA (at least in the user point of view). As shown in Figure 130, the first desired candidate is ranked 37th in the list. In fact, the order of the list is more accurate when the IA is supplement with knowledge of NLP-tool. This will be illustrated later in this section.

A promising result is Recall (for service individual) of the list of composition candidates. The recall is equal to 100 %. This means that we have 7 relevant services retrieved (*TheaterSearch*, *CITYTheaterSearch*, *PRICETheaterSearch*, *STATETheaterSearch*, *MailDaemon*, *MessagingServer*, *GlobalSMS*) and there are only these seven services in the service repository that are relevant to the request. This indicates the highest performance possible for this scenario.

Secondly, IA processes a request by using the keyword search, ontology relation and knowledge from NLP-tool (the MRS functionality is enabled). The composition candidates are shown in Figure 131. While the MRS functionality is enabled, IA validates the composition candidates in Figure 130 with NLP-tool result. The composition candidates that do not agree with the NLP-tool result will be degraded (but not removed). Hence, the precision index and recall are the same as those of Figure 130. However, the ranking of the service candidates is more accurate. Essentially, the first desired candidate is ranked 1st in the list.

Query response time for processing the request “Find an address of theater in my area and send it to my email” (and MRS is not utilized) is equal to 49328 ms. In other words, IA requires 49328 ms to process the request (i.e. derive and rank service composition candidates) giving that there are 141 services in the repository. Query response time for processing the request “Find an address of theater in my area and send it to my email” (and MRS is utilized) is equal to 113125 ms. In other words, IA requires 113125 ms to process the request (i.e. derive and rank service composition candidates) giving that there are 141 services in the repository. Hence, IA requires 63797 ms (i.e. 113125 - 49328) more to process the request if the MRS is utilized. This brings about 129 % increased in the execution time if MRS is utilized. This increment is mainly caused by the standard I/O operation (i.e. The MRS of user request is stored in an XML file. IA parses and analyzes the XML file when the MRS feature is turned on). In the future work (where the MRS functionality is integrated properly (e.g. IA does not need the MRS result from external file)), we expect an improvement on the response time. This is based on an assumption that the efficiency of the grammar and efficiency of the parses are the same as before.

```
++List of Composition Candidates sorted by total weight++
[<MailDaemon,0>, <TheaterSearch,50>, <UKLocation,14>, <GetRestaurantByAddress,7>]
[<MailDaemon,0>, <TheaterSearch,50>, <UKLocationPro,14>, <GetRestaurantByAddress,5>]
[<MailDaemon,0>, <UKLocation,50>, <GetRestaurantByAddress,7>, <CITYTheaterSearch,16>]
[<MailDaemon,0>, <UKLocation,50>, <GetRestaurantByAddress,7>, <PRICETheaterSearch,16>]
[<MailDaemon,0>, <UKLocation,50>, <GetRestaurantByAddress,7>, <STATETheaterSearch,16>]
[<MailDaemon,0>, <UKLocationPro,50>, <GetRestaurantByAddress,5>, <CITYTheaterSearch,16>]
[<MailDaemon,0>, <UKLocationPro,50>, <GetRestaurantByAddress,5>, <PRICETheaterSearch,16>]
[<MessagingServer,0>, <TheaterSearch,50>, <UKLocation,14>, <GetRestaurantByAddress,7>]
[<MailDaemon,0>, <UKLocationPro,50>, <GetRestaurantByAddress,5>, <STATETheaterSearch,16>]
[<GlobalSMS,0>, <TheaterSearch,50>, <UKLocation,14>, <GetRestaurantByAddress,7>]
[<MessagingServer,0>, <TheaterSearch,50>, <UKLocationPro,14>, <GetRestaurantByAddress,5>]
```


Result and Discussion

```
[<UKLocation,0>, <GetRestaurantByAddress,7>, <CITYTheaterSearch,16>]
[<UKLocation,0>, <GetRestaurantByAddress,7>, <PRICETheaterSearch,16>]
[<UKLocation,0>, <GetRestaurantByAddress,7>, <STATETheaterSearch,16>]
[<UKLocationPro,0>, <GetRestaurantByAddress,5>, <CITYTheaterSearch,16>]
[<UKLocationPro,0>, <GetRestaurantByAddress,5>, <STATETheaterSearch,16>]
[<UKLocationPro,0>, <GetRestaurantByAddress,5>, <PRICETheaterSearch,16>]
[<MailDaemon,0>, <UKLocation,50>, <GetRestaurantByAddress,7>]
[<MailDaemon,0>, <UKLocationPro,50>, <GetRestaurantByAddress,5>]
[<GetRestaurantByAddress,0>, <CITYTheaterSearch,16>]
[<GetRestaurantByAddress,0>, <PRICETheaterSearch,16>]
[<GetRestaurantByAddress,0>, <STATETheaterSearch,16>]
[<TheaterSearch,0>, <UKLocation,14>]
[<TheaterSearch,0>, <UKLocationPro,14>]
[<MailDaemon,0>, <URLToIPAddress,5>, <MailLocate,16>]
[<GlobalSMS,0>, <UKLocation,50>, <GetRestaurantByAddress,7>]
[<MessagingServer,0>, <UKLocation,50>, <GetRestaurantByAddress,7>]
[<MessagingServer,0>, <UKLocationPro,50>, <GetRestaurantByAddress,5>]
[<GlobalSMS,0>, <UKLocationPro,50>, <GetRestaurantByAddress,5>]
[<UKLocation,0>, <CITYTheaterSearch,14>]
[<MailDaemon,0>, <GetRestaurantByAddress,33>]
[<UKLocation,0>, <STATETheaterSearch,14>]
[<UKLocation,0>, <PRICETheaterSearch,14>]
[<TheaterSearch,0>]
[<MessagingServer,0>, <URLToIPAddress,5>, <MailLocate,16>]
[<GlobalSMS,0>, <URLToIPAddress,5>, <MailLocate,16>]
[<UKLocationPro,0>, <CITYTheaterSearch,1>]
[<UKLocationPro,0>, <PRICETheaterSearch,1>]
[<UKLocationPro,0>, <STATETheaterSearch,1>]
[<MailDaemon,0>, <UKLocation,50>]
[<MailDaemon,0>, <UKLocationPro,50>]
[<MailDaemon,0>, <MailLocate,20>]
[<GlobalSMS,0>, <GetRestaurantByAddress,33>]
[<MessagingServer,0>, <GetRestaurantByAddress,33>]
[<CITYTheaterSearch,0>]
[<PRICETheaterSearch,0>]
[<STATETheaterSearch,0>]
[<MailDaemon,0>, <URLToIPAddress,5>]
[<MessagingServer,0>, <UKLocation,50>]
[<MessagingServer,0>, <UKLocationPro,50>]
[<GlobalSMS,0>, <UKLocation,50>]
[<GlobalSMS,0>, <UKLocationPro,50>]
[<MessagingServer,0>, <MailLocate,20>]
[<GlobalSMS,0>, <URLToIPAddress,5>]
[<URLToIPAddress,0>, <MailLocate,16>]
[<GlobalSMS,0>, <MailLocate,5>]
[<MessagingServer,0>, <URLToIPAddress,5>]
[<MailDaemon,0>]
[<GetRestaurantByAddress,0>, <UKLocationPro,14>]
[<UKLocation,0>, <GetRestaurantByAddress,7>]
[<MessagingServer,0>]
[<GlobalSMS,0>]
[<GetRestaurantByAddress,0>]
[<MailLocate,0>]
[<URLToIPAddress,0>]
[<SkyScanner,0>]
[<UKLocation,0>]
[<UKLocationPro,0>]
end ++List of Composition Candidates sorted by total weight++
```

Figure 130: List of Composition Candidates for scenario 3 (MRS disabled)

Result and Discussion

```
++List of Composition Candidates sorted by nlp++
[<MailDaemon,0>, <TheaterSearch,50>]
[<MailDaemon,0>, <CITYTheaterSearch,50>]
[<MailDaemon,0>, <PRICETheaterSearch,50>]
[<GlobalSMS,0>, <TheaterSearch,50>]
[<MailDaemon,0>, <STATETheaterSearch,50>]
[<MessagingServer,0>, <TheaterSearch,50>]
[<GlobalSMS,0>, <CITYTheaterSearch,50>]
[<MessagingServer,0>, <CITYTheaterSearch,50>]
[<GlobalSMS,0>, <PRICETheaterSearch,50>]
[<MessagingServer,0>, <PRICETheaterSearch,50>]
[<GlobalSMS,0>, <STATETheaterSearch,50>]
[<MessagingServer,0>, <STATETheaterSearch,50>]
[<GetRestaurantByAddress,0>, <CITYTheaterSearch,16>]
[<GetRestaurantByAddress,0>, <PRICETheaterSearch,16>]
[<GetRestaurantByAddress,0>, <STATETheaterSearch,16>]
[<UKLocation,0>, <CITYTheaterSearch,14>]
[<MailDaemon,0>, <GetRestaurantByAddress,33>]
[<UKLocation,0>, <STATETheaterSearch,14>]
[<UKLocation,0>, <PRICETheaterSearch,14>]
[<UKLocationPro,0>, <CITYTheaterSearch,1>]
[<UKLocationPro,0>, <PRICETheaterSearch,1>]
[<UKLocationPro,0>, <STATETheaterSearch,1>]
[<MailDaemon,0>, <UKLocation,50>]
[<MailDaemon,0>, <UKLocationPro,50>]
[<MailDaemon,0>, <MailLocate,20>]
[<GlobalSMS,0>, <GetRestaurantByAddress,33>]
[<MessagingServer,0>, <GetRestaurantByAddress,33>]
[<MailDaemon,0>, <URLToIPAddress,5>]
[<MessagingServer,0>, <UKLocation,50>]
[<MessagingServer,0>, <UKLocationPro,50>]
[<GlobalSMS,0>, <UKLocation,50>]
[<GlobalSMS,0>, <UKLocationPro,50>]
[<MessagingServer,0>, <MailLocate,20>]
[<GlobalSMS,0>, <URLToIPAddress,5>]
[<GlobalSMS,0>, <MailLocate,5>]
[<MessagingServer,0>, <URLToIPAddress,5>]
[<GetRestaurantByAddress,0>, <UKLocationPro,14>]
[<UKLocation,0>, <GetRestaurantByAddress,7>]
[<MailDaemon,0>, <TheaterSearch,50>, <UKLocation,14>, <GetRestaurantByAddress,7>]
[<MailDaemon,0>, <TheaterSearch,50>, <UKLocationPro,14>, <GetRestaurantByAddress,5>]
[<MailDaemon,0>, <UKLocation,50>, <GetRestaurantByAddress,7>, <CITYTheaterSearch,16>]
[<MailDaemon,0>, <UKLocation,50>, <GetRestaurantByAddress,7>, <PRICETheaterSearch,16>]
[<MailDaemon,0>, <UKLocation,50>, <GetRestaurantByAddress,7>, <STATETheaterSearch,16>]
[<MailDaemon,0>, <UKLocationPro,50>, <GetRestaurantByAddress,5>, <CITYTheaterSearch,16>]
[<MailDaemon,0>, <UKLocationPro,50>, <GetRestaurantByAddress,5>, <PRICETheaterSearch,16>]
[<MessagingServer,0>, <TheaterSearch,50>, <UKLocation,14>, <GetRestaurantByAddress,7>]
[<MailDaemon,0>, <UKLocationPro,50>, <GetRestaurantByAddress,5>, <STATETheaterSearch,16>]
[<GlobalSMS,0>, <TheaterSearch,50>, <UKLocation,14>, <GetRestaurantByAddress,7>]
[<MessagingServer,0>, <TheaterSearch,50>, <UKLocationPro,14>, <GetRestaurantByAddress,5>]
[<GlobalSMS,0>, <TheaterSearch,50>, <UKLocationPro,14>, <GetRestaurantByAddress,5>]
[<GlobalSMS,0>, <UKLocation,50>, <GetRestaurantByAddress,7>, <CITYTheaterSearch,16>]
[<MessagingServer,0>, <UKLocation,50>, <GetRestaurantByAddress,7>, <CITYTheaterSearch,16>]
[<GlobalSMS,0>, <UKLocation,50>, <GetRestaurantByAddress,7>, <STATETheaterSearch,16>]
[<MessagingServer,0>, <UKLocation,50>, <GetRestaurantByAddress,7>, <STATETheaterSearch,16>]
[<GlobalSMS,0>, <UKLocation,50>, <GetRestaurantByAddress,7>, <PRICETheaterSearch,16>]
[<MessagingServer,0>, <UKLocation,50>, <GetRestaurantByAddress,7>, <PRICETheaterSearch,16>]
[<MessagingServer,0>, <UKLocationPro,50>, <GetRestaurantByAddress,5>, <CITYTheaterSearch,16>]
[<GlobalSMS,0>, <UKLocationPro,50>, <GetRestaurantByAddress,5>, <CITYTheaterSearch,16>]
```

Result and Discussion

```
[<MessagingServer,0>, <UKLocationPro,50>, <GetRestaurantByAddress,5>, <PRICETheaterSearch,16>]
[<GlobalSMS,0>, <UKLocationPro,50>, <GetRestaurantByAddress,5>, <PRICETheaterSearch,16>]
[<MessagingServer,0>, <UKLocationPro,50>, <GetRestaurantByAddress,5>, <STATETheaterSearch,16>]
[<GlobalSMS,0>, <UKLocationPro,50>, <GetRestaurantByAddress,5>, <STATETheaterSearch,16>]
[<MailDaemon,0>, <TheaterSearch,50>, <UKLocationPro,14>]
[<MailDaemon,0>, <TheaterSearch,50>, <UKLocation,14>]
[<MailDaemon,0>, <GetRestaurantByAddress,33>, <CITYTheaterSearch,16>]
[<MailDaemon,0>, <GetRestaurantByAddress,33>, <PRICETheaterSearch,16>]
[<MailDaemon,0>, <GetRestaurantByAddress,33>, <STATETheaterSearch,16>]
[<MailDaemon,0>, <UKLocation,50>, <CITYTheaterSearch,14>]
[<MessagingServer,0>, <TheaterSearch,50>, <UKLocationPro,14>]
[<MailDaemon,0>, <UKLocation,50>, <PRICETheaterSearch,14>]
[<GlobalSMS,0>, <TheaterSearch,50>, <UKLocationPro,14>]
[<GlobalSMS,0>, <TheaterSearch,50>, <UKLocation,14>]
[<MailDaemon,0>, <UKLocation,50>, <STATETheaterSearch,14>]
[<MessagingServer,0>, <TheaterSearch,50>, <UKLocation,14>]
[<MessagingServer,0>, <GetRestaurantByAddress,33>, <CITYTheaterSearch,16>]
[<GlobalSMS,0>, <GetRestaurantByAddress,33>, <CITYTheaterSearch,16>]
[<MessagingServer,0>, <GetRestaurantByAddress,33>, <PRICETheaterSearch,16>]
[<GlobalSMS,0>, <GetRestaurantByAddress,33>, <PRICETheaterSearch,16>]
[<MessagingServer,0>, <GetRestaurantByAddress,33>, <STATETheaterSearch,16>]
[<GlobalSMS,0>, <GetRestaurantByAddress,33>, <STATETheaterSearch,16>]
[<MailDaemon,0>, <UKLocationPro,50>, <CITYTheaterSearch,1>]
[<MailDaemon,0>, <UKLocationPro,50>, <PRICETheaterSearch,1>]
[<MailDaemon,0>, <UKLocationPro,50>, <STATETheaterSearch,1>]
[<MessagingServer,0>, <UKLocation,50>, <CITYTheaterSearch,14>]
[<GlobalSMS,0>, <UKLocation,50>, <CITYTheaterSearch,14>]
[<GlobalSMS,0>, <UKLocation,50>, <STATETheaterSearch,14>]
[<GlobalSMS,0>, <UKLocation,50>, <PRICETheaterSearch,14>]
[<MessagingServer,0>, <UKLocation,50>, <PRICETheaterSearch,14>]
[<MessagingServer,0>, <UKLocation,50>, <STATETheaterSearch,14>]
[<MessagingServer,0>, <UKLocationPro,50>, <CITYTheaterSearch,1>]
[<GlobalSMS,0>, <UKLocationPro,50>, <CITYTheaterSearch,1>]
[<MessagingServer,0>, <UKLocationPro,50>, <STATETheaterSearch,1>]
[<GlobalSMS,0>, <UKLocationPro,50>, <STATETheaterSearch,1>]
[<MessagingServer,0>, <UKLocationPro,50>, <PRICETheaterSearch,1>]
[<GlobalSMS,0>, <UKLocationPro,50>, <PRICETheaterSearch,1>]
[<TheaterSearch,0>, <UKLocation,14>, <GetRestaurantByAddress,7>]
[<TheaterSearch,0>, <UKLocationPro,14>, <GetRestaurantByAddress,5>]
[<UKLocation,0>, <GetRestaurantByAddress,7>, <CITYTheaterSearch,16>]
[<UKLocation,0>, <GetRestaurantByAddress,7>, <PRICETheaterSearch,16>]
[<UKLocation,0>, <GetRestaurantByAddress,7>, <STATETheaterSearch,16>]
[<UKLocationPro,0>, <GetRestaurantByAddress,5>, <CITYTheaterSearch,16>]
[<UKLocationPro,0>, <GetRestaurantByAddress,5>, <STATETheaterSearch,16>]
[<UKLocationPro,0>, <GetRestaurantByAddress,5>, <PRICETheaterSearch,16>]
[<MailDaemon,0>, <UKLocation,50>, <GetRestaurantByAddress,7>]
[<MailDaemon,0>, <UKLocationPro,50>, <GetRestaurantByAddress,5>]
[<TheaterSearch,0>, <UKLocation,14>]
[<TheaterSearch,0>, <UKLocationPro,14>]
[<MailDaemon,0>, <URLToIPAddress,5>, <MailLocate,16>]
[<GlobalSMS,0>, <UKLocation,50>, <GetRestaurantByAddress,7>]
[<MessagingServer,0>, <UKLocation,50>, <GetRestaurantByAddress,7>]
[<MessagingServer,0>, <UKLocationPro,50>, <GetRestaurantByAddress,5>]
[<GlobalSMS,0>, <UKLocationPro,50>, <GetRestaurantByAddress,5>]
[<TheaterSearch,0>]
[<MessagingServer,0>, <URLToIPAddress,5>, <MailLocate,16>]
[<GlobalSMS,0>, <URLToIPAddress,5>, <MailLocate,16>]
[<CITYTheaterSearch,0>]
[<PRICETheaterSearch,0>]
[<STATETheaterSearch,0>]
```

Result and Discussion

```
[<URLToIPAddress,0>, <MailLocate,16>]
[<MailDaemon,0>]
[<MessagingServer,0>]
[<GlobalSMS,0>]
[<GetRestaurantByAddress,0>]
[<MailLocate,0>]
[<URLToIPAddress,0>]
[<SkyScanner,0>]
[<UKLocation,0>]
[<UKLocationPro,0>]
end ++List of Composition Candidates sorted by nlp++
```

Figure 131: List of Composition Candidates for scenario 2 (MRS enabled)

We also evaluated the IA performance on the context-handling functionality (how well IA satisfies the user proposition as defined in ContextRule.txt). To accomplish user scenario 3, as discussed in section 5.1.3, the end user defines a service composition to be invoked when the certain user location (e.g. customer_siteA@telin.nl , customer_siteB@telin.nl) has been changed. If the user location has been changed to customer_siteA@telin.nl (the user is at customer site A), IA constructs a service composition as shown in Figure 132. It is easily seen that the composition and the input instances for service components (e.g. customer_siteA@telin.nl and customer_siteB@telin.nl are input instances of DistanceService) are exactly the same with what is defined in ContextRule.txt. Thus, IA performance (with regards to preciseness of service composition) is 100 %. Clearly, this is because the user manually states the service composition and its service input parameters. A similar service composition is composed when a user location is changed to customer_siteB@telin.nl, customer_siteC@telin.nl or customer_siteD@telin.nl with appropriate service input parameters (e.g. customer_siteB@telin.nl and customer_siteC@telin.nl are input instances of DistanceService when user location has been changed to customer_siteB@telin.nl).

If the user location has been changed to customer_siteE@telin.nl (the user is at the customer site E, which is the last place he planned to visit), IA constructs a service composition as shown in Figure 133. It is easily seen that the composition and the input instance (e.g. deal.doc is an input instance of MessagingServer) for service components are exactly the same with what is defined in ContextRule.txt. Thus, IA performance (with regards to preciseness of service composition) is 100 %. Clearly, this is because the user manually states the service composition and its service input parameters.

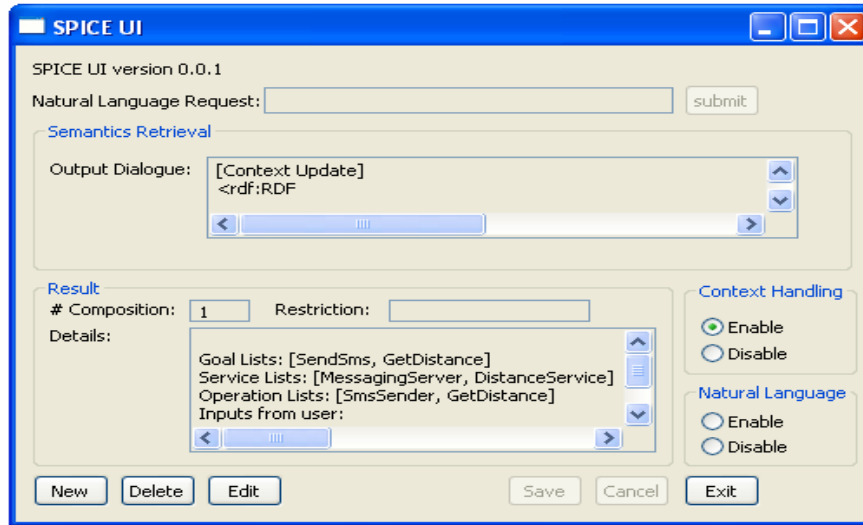


Figure 132: IA constructs a service composition when a user context has been changed to customer_siteA@telin.nl

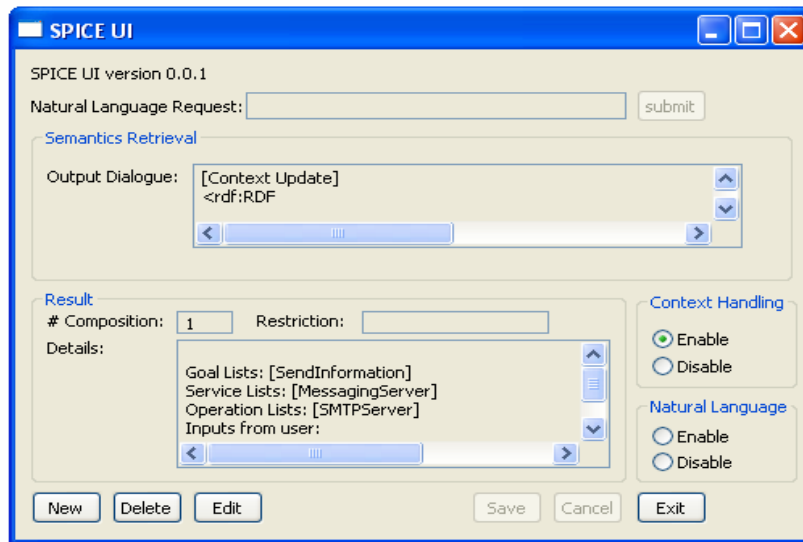


Figure 133: IA constructs a service composition when a user context has been changed to customer_siteE@telin.nl

5.5 A Performance Improvement with MRS

We have seen in section 5.4 that IA performance can be improved with MRS of the user request. It is more accurate to rank a list of service composition candidates if IA is supplement with knowledge of NLP-tool. For example, as shown in Figure 122, IA that runs without MRS of user request ranks the service candidates inappropriately. The service [$\langle \text{SmsSender}, 0 \rangle, \langle \text{UKLocation}, 50 \rangle, \langle \text{VEGRestaurantFinding}, 8 \rangle$] is the first one in the rank instead of the more appropriate candidate, [$\langle \text{MessagingServer}, 0 \rangle, \langle \text{VEGRestaurantFinding}, 33 \rangle$]. On the other hand, IA that runs with MRS ranks

Result and Discussion

the service candidates more correctly (i.e. the service candidate, [*<MessagingServer,0>*,*<VEGRestaurantFinding,33>*] is the first one in the rank).

Without MRS, it is likely that the service compositions, that are composed of many service components, have higher rank than the service compositions, that are composed of few service components. This is because IA ranks composition candidates by using a sum of total goal weight (i.e. it is likely that the candidate, that has many service goals, has total goal weight greater than the candidate that has fewer service goals) and total connection weight (i.e. it is likely that the candidate, that is composed of many service components, has total connection weight (how well one service component links with another adjacent service component) greater than the candidate that is composed of fewer service components).

With MRS, the rank of composition candidate is adjusted so that composition candidates that have a number of service components agrees with a user request get higher rank than other candidates. Furthermore, the composition candidates that are concurred with MRS of user request get higher rank than other candidates. Suppose that there are three service candidates (for the user request “Find vegetarian restaurant in my area and send it by sms or email”). The first service candidate is [*<SmsSender,0>*, *<UKLocation,50>*, *<VEGRestaurantFinding,8>*]. The second service candidate is [*<MessagingServer,0>*, *<VEGRestaurantFinding,33>*]. The third service candidate is [*<MailDaemon,0>*, *<VEGRestaurantFinding,33>*]. Suppose also that, without MRS, IA ranks the three service candidates as:

```
[<SmsSender,0>, <UKLocation,50>, <VEGRestaurantFinding,8>]
[<MessagingServer,0>, <VEGRestaurantFinding,33>]
[<MailDaemon,0>, <VEGRestaurantFinding,33>]
```

With an analysis on MRS of user request, IA realizes that there are two service components per one service composition (that is “find vegetarian restaurant” and “send sms” or “send email”). Furthermore, an analysis on MRS reveals that an output of a service, *VEGRestaurantFinding*, could be used as an input of another service, *MessagingServer*. Moreover, an output of a service, *VEGRestaurantFinding*, could be used as an input of another service, *MailDaemon*. With this knowledge, IA ranks the list of service composition candidates as:

```
[<MessagingServer,0>, <VEGRestaurantFinding,33>]
[<MailDaemon,0>, <VEGRestaurantFinding,33>]
[<SmsSender,0>, <UKLocation,50>, <VEGRestaurantFinding,8>]
```

An analysis on the MRS can also be used to improve a correctness of input parameter matching. Specifically, the input instances that do not agree with the analysis of MRS are taken off. For example, in the request “Search a flight from London to Paris on 10-06-2008 and book it”, we get (the IA is run without knowledge of MRS):

```
Inputs from
user[<FlightBooking,BookFlight>,[<10-06-2008,9>]]
[<SkyScanner,SearchFlight>,[<London,9>, <Paris,9>, <London,9>, <Paris,9>, <10-06-2008,9>]]
```

If the MRS of user request is deployed, we get:

Result and Discussion

Inputs from user[<SkyScanner,SearchFlight>,[<London,9>, <Paris,9>, <10-06-2008,9>]][<FlightBooking,BookFlight>,[<10-06-2008,9>]]

In the result (without MRS knowledge), it indicates that the departure city can be either London or Paris and the arrival city can be either London or Paris. In the result (with MRS knowledge), it is more accurate to indicate that the departure city can be only London and the arrival city can be only Paris (the preposition “from” and “to” are used to emphasize on this fact).

In section 5.4, we present query response times needed for IA in order to process a user request. The query response times are calculated in two scenarios, namely 1) when the MRS is not utilized and 2) when the MRS is utilized. Normally, the IA needs more time to process a user request if the MRS is utilized. However, the response time is increased slightly (in a magnitude of millisecond). Therefore, the performance gains (in term of accuracy), when MRS is utilized, is considered to be higher than the performance loss (in term of speed).

Chapter 6

Conclusion and Future work

In this chapter, we provide a conclusion of this thesis. We will describe how well the objective (requirement) of this thesis has been accomplished. In the end of this chapter, we provide a discussion about the possibilities to extend/improve our works.

6.1 Conclusion

This thesis covers two fields, namely, computer science (which relates to Natural Language Processing) and telecommunication (which relates to context handling, semantic web technologies, service discovery and composition). In this thesis, we have developed a search and deploy application (i.e. IA) that receives a request from end user in natural language (English), analyze the request, search and compose for service composition candidates that satisfy the request and deploy the candidates into an artificial language, SPATEL [43].

From a request submitted by the end user, IA does not only find services in the repository that satisfy the request but also figures out how to compose those services (if there are more than one service) based on semantics of the service parameters (i.e. service input and service output) and a semantic of the request. Furthermore, IA associates words in the request with the service input parameters based on semantics of the service inputs and a semantic of the request. For example, if a user request is “find a weather report in Paris and send it to my email or my mobile phone”, IA will take “Paris” as an input of one service (e.g. WeatherReport). Moreover, IA will figure out that “my email” is associated with the user’s email (e.g. Remco@telin.nl) which is an input of another service (e.g. SendEmail).

A context handling feature has also been considered in this thesis. We have developed a module of IA that registers and subscribes for a user context published by a knowledge source in KMF framework. IA takes into account a context of user (e.g. user location, user authentication) when it selects services from the service repository. For example, if user is driving (i.e. the user is in the car) and IA needs to send information to the user, a service, SendEmail, is not an appropriate service (e.g. it is not plausible for the user to check email while he is driving),

Conclusion and Future work

rather, a service, SendSMS is more appropriate. IA also ensures that the desired service composition will be deployed when certain condition on user context arises.

Moreover, a service operator can restrict an access of user to set of services via restriction/authentication context. When IA receives a restriction context (in practice, IA should process the restriction context even if the context handling feature is turned off), IA will limit an access of user to the specified service in the context. When IA receives an authentication context (in practice, IA should process the authentication context even if the context handling feature is turned on), IA will grant an access of user to the specified service in the context.

Firstly, we have developed modules/components (e.g. information extraction, input searching, service discovery, service composition, sorter, ACE-interface defined in section 1.3) to support service discovery/composition based on keyword search and ontology relation. We have created three ontology files, namely, goal ontology, input/output ontology and synonym ontology. Goal ontology is used to determine significance (i.e. an influence) of keywords in the user request. In other words, it is used to determine how well the keywords correspond to the service goals in the ontology. For example, a keyword “by” alone is not enough to justify that the service that has service goal, “FindRestaurantByLocation” and the service that has service goal, “SendInformationByEmail” should be selected. This is because the keyword “by” corresponds to two distinct service goals (FindRestaurantByLocation and SendInformationByEmail).

Therefore, it depends on existences of other keywords (e.g. “find” , “restaurant”, “location”) for the services (e.g. FindRestaurantByLocation) to be selected. On the other hand, a keyword “sas” is sufficient to justify that the service that has service goal, “SearchSasFlight” should be selected. This is because the keyword “sas” only corresponds to one service goal (SearchSasFlight) in the goal ontology.

Input/output ontology is used to connect the service components during a process of service composition. It is also used to parameterize the service candidates with keywords found in the request. Synonym ontology is used to identify words that have the same meanings (e.g. “find” and “search”). We do not put restriction on the user request. The words in user request can be varied as long as it is defined in Synonym ontology (e.g. a user can request either “find restaurant in Paris” or “search restaurant in Paris”).

We have also created a service repository file (composed of 141 services) for our demonstration purpose. IA takes all keywords from the user request, search for services (that correspond to the keywords) from the repository and compose these services together. If more than one service compositions are created, IA will rank the composition candidates based on a degree in which the candidate satisfies the user request and a degree in which the components of the candidate are connected together.

The composition candidates will then be exported into SPATEL. The candidates that are exported into SPATEL are ready to be executed by SPATEL execution engine providing that the appropriate service inputs (extracted from the request) will also be passed to the engine. At the time of writing this thesis, the SPATEL execution engine is not fully function (it still needs to be installed and run manually). Therefore, we have not verified the SPATEL generated by our application with the engine, but rather, we verify the format of the SPATEL files and the input taken from the user request.

Secondly, we have developed a module/component (e.g. knowledge sink defined in section 1.3) to support context-handling. We have created two files: recommended service rule and restriction service rule. The recommended service rule contains a table indicating service composition that should be executed when the user moves to certain locations. The restriction service rule contains a table indicating service that should be restricted (i.e. access denied) when the user moves to certain locations. The context handling module ensures that rules defined in these two files are enforced. Moreover, the context handling module monitors user authentication based on restriction/authentication context. In fact, the context handling module will process on user context if and only if 1) it is an update context (e.g. timestamp) and 2) the context belongs to that particular user.

Thirdly, we have improved preciseness on service compositions by using MRS obtained from linguistic expert. The MRS represents a connection between words in the user request. From MRS, we can derive an exact number of service components and get a clue on how the service components are composed together. We can also infer that which words in the request should be used as input instances of which services. However, the MRS(s) that we obtained are in Norwegian (the application that generates the MRS supports request in Norwegian). Nevertheless, for a simple request (as in our scenarios), it is straightforward to translate the MRS into English (using a word-to-word translation). To achieve this, we have created a dictionary file, `norwegian2english.txt`. The natural language processing module utilizes the dictionary file in order to translate the MRS into English. Hence, the translated MRS corresponds to the request that the user handed over to IA. The natural language processing module validates the service composition candidates and the service inputs with the MRS. The list of composition candidates are rearranged such that the composition candidates that agree with MRS gets higher rank than the other candidates. Moreover, the service inputs that do not agree with MRS are removed. Consequently, a performance of natural language processing is improved.

Fourth, we have experimented with the IA by inventing three user scenarios. We have tested, among other things, natural language processing competency, service discovery and service composition capability. We have also tested the context handling capability. Furthermore, we have measured the IA discovery performance by using precision index and recall. In average, the precision index to discover individual services is equal to the traditional keyword-based ways of service discovery. However, the precision index to discover service compositions is incomparable to the traditional keyword-based ways of service discovery. This is because the idea of IA is to propose all possible individual service candidates (and all possible service composition candidates that are composed from the service candidates) to the end user. In other words, it is better to propose too many service candidates to the end user rather than too few service candidates (in which the desired service candidates may not be found).

One disadvantage of having low precision index is that there would be too many services that are not useful proposed to the user. We have shown that this problem can be alleviated by ranking the service candidates with MRS. Thus, it is more likely that the suitable candidates are easily found in the list (i.e. they are the first ones proposed to the user). A promising result is Recall. For every scenarios, the recall is equal to 100 %. This indicates the highest performance possible (and higher than the traditional keyword-based ways of service discovery). This indicates that all plausible services (that are suitable to the user request) in the repository are selected by IA. As for a query response time, we have shown that the user requests are processed in a magnitude of milliseconds. Thus, this indicates an acceptable performance. Moreover, we have shown that the query response time will be increased if IA utilizes MRS. We noticed that this could be caused by an IO operation (e.g. the MRS file is read by IA). However, we believe

that the performance gain in term of preciseness (if MRS file is used) is better than the performance loss in term of speed.

In conclusion, we have demonstrated possibilities of introducing an end user studio (i.e. IA) that helps facilitating service compositions. The end user studio embraces two features: natural language processing feature and context handling feature. Although there was a work on natural language processing in SPICE project, the work is limited to bounded domains as they depend on structure of sentence (i.e. user request). Our work in processing the natural language request is based on keyword-based discovery and semantic relation. Thus, it does not depend on structure of sentence (it is not limited to bounded domains). Moreover, the meaning of user request is not overlooked as we used the MRS (provided by linguistic expert) to validate our service candidates. In other words, the keyword-based discovery is used to bring in all plausible services that correspond to keyword in user request. The service composition candidates are then constructed (and ranked) based on semantic relation of the service components. In fact, some constructed service composition candidates do not agree with the meaning of user request. To improve on the performance, the MRS is then used to tailor the list of service candidate (i.e. the candidates that do not agree with the request are given lower ranks than the others).

Our work also demonstrates how the user contexts could be utilized by the end user studio. We have employed the existing technology developed in SPICE project to publish and subscribe the contexts. Based on the contexts, the end user studio decides on which services should be returned to the end user. Furthermore, we have experimented with three user scenarios. The three user scenarios illustrate how we can benefit from natural language processing and context handling functionalities in real-life. However, it is worth noting that our work still has room for further improvement. Among other things, in future works, we should consider to process the conditional sentence and to improve on the performance with regards to query response time and precision index.

6.2 Future work

In this section, we discuss about possibilities to enhance our works. The enhancements, that are directly related to our design decision, have been discussed in chapter 3 and we will highlight them again in this section. Moreover, we will discuss on other enhancements, that are related to interactions between the components in the architecture, that is proposed in our project.

Firstly, we could introduce a centralized database to keep a service repository. Then, IA could interact with this database by using web service technology (e.g. exchanging SOAP message). An advantage of the future approach would be that it alleviates maintenance problem. For example, if a new service were to be added to the repository, it could have been added directly to the database. Otherwise, every service repository files would need to be updated (recall that each end user has a service repository file stored locally).

Secondly, we could also introduce a centralized database to keep all necessary ontologies (e.g. Goal ontology, Input/Output ontology, Synonym ontology). Then, IA could interact with this database by using web service technology (e.g. exchanging SOAP message). An advantage of the future approach would be that it alleviates maintenance problem. For example, if a new service goal were to be added to the repository, it could have been added directly to the database. Otherwise, every ontology files would need to be updated (recall that each end user has ontology files stored locally).

Thirdly, in the natural language processing module, we could handle a conditional expression (e.g. if, else, otherwise etc). This is the biggest challenge. To support this function, the NLP-tool (employed by IA) should be able to extract the conditional predicate. Moreover, the IA should be able to reason about (i.e. evaluate) the conditional predicate. Suppose that a user request is “If the temperature in Paris is less than twenty degree, send me the weather report in Paris by email”. To process this request, IA would employ NLP-tool to analyze the request and generate the MRS of the sentence. From MRS, IA recognizes that the user request, basically, is composed of a conditional expression in a form of [if<sentence1>-><sentence2>]. Specifically, IA knows that it should evaluate the first sentence (i.e. temperature in Paris is less than twenty degree) and process the second sentence only if the evaluation value is ‘true’. In order to evaluate the first sentence, IA needs to have some knowledge on the subject (i.e. temperature in Paris). If IA were to have the knowledge on the subject, it could have checked its knowledge with the condition set by the user (i.e. less than twenty degree). If the condition was satisfied, IA could process the second sentence as usual (that is to find service to be invoked). Otherwise, the IA would ignore the user request.

However, if IA does not have knowledge on the subject (i.e. temperature in Paris), it would have created the knowledge from list of services (or even compose appropriate service compositions for this purpose) that are available (that is to find temperature in Paris). Then, the IA could have checked the derived knowledge with the condition set by the user. If the condition was satisfied, IA could process the second sentence as usual (that is to find service to be invoked). Otherwise, the IA would ignore the user request.

Even if the user request were more sophisticated (e.g. if<sentence1>->if<sentence2>-><sentence3>), the IA could have handled it by applying the above process, recursively. An ability to process and analyze the conditional expressions would make the application more flexible (and more helpful) in processing the user request. The end user could tailor services (that will be delivered by IA) by using the set of conditional statement (e.g. “If the temperature in Paris is more than twenty degree, search outdoor activities, otherwise search indoor activities”). Unfortunately, due to time limitations, this functionality has not been included in the current work. However, the functionality should be covered in the future work as we believe that it will greatly increase an IA ability to process the natural language request.

Fourth, the context-handling ability of IA could be enhanced. Currently, IA does not take a **Quality of Context (QoC)** [35] into account. In reality, the contexts have different quality depending on types of context source. Moreover, the context information provided by different context sources may be conflicted. To improve on this, IA should select the context based on QoC. In particular, IA should select the context that has highest quality and ignore those of conflicted context information that has lower quality.

Fifth, the interaction between IA and NLP-tool has been done manually (we take the MRS file from NLP-tool and feed it to IA, manually). This is because the compatibility issue between IA and NLP-tool (i.e. IA developed in Java supports English request, NLP-tool developed in Prolog supports Norwegian request). We could improve on this by changing both applications to support the same language (either English or Norwegian) and deploy NLP-tool, for example, as a web service, in which, the IA can interact with (i.e. via SOAP message), automatically.

Sixth, for an ambiguous sentence such as “Find vegetarian restaurant in my area and send it by email”, IA had a difficulty to deduce some service inputs. This is because the user request is

Conclusion and Future work

not complete. IA could not infer an owner of the email from the sentence (i.e. “email” does not explicitly state who is the owner of the email). If the request were to be “Find vegetarian restaurant in my area and send it to **my** email”, IA would have been able to extract the service input of MailDaemon. Indeed, this illustrates a limitation of IA for an ambiguous sentence. IA would have guessed from the context of the sentence that the owner of email is the user, himself. This could be a challenge in the future work.

Lastly, at the time of writing this thesis, the SPATEL execution engine is unfortunately not ready for automatic execution (we have to execute the SPATEL execution engine manually using 1) the SPATEL files and 2) the input parameter derived from user request) and the web service used for our demonstration purpose does not physically exist. Hence, we could not obtain the execution result from the SPATEL execution engine.

It is worth noting that the study on SPATEL and SPATEL execution engine are conducted in other research projects and it is beyond the scope of this thesis. As aforementioned, an objective of this thesis is to demonstrate on service discovery and composition based on the natural language request (and some user contexts). Therefore, it is sufficient to achieve the objective by showing service compositions (and the degree in which it satisfies the user request). Nevertheless, a real execution on the service compositions produced by IA is motivated and should be included in the future work.

Appendix A

IA Algorithm and APIs

One of technical difficulties found during the implementation is to realize ideas proposed in chapter 3. In this section, we present six complex algorithms that have been developed during the project. The first algorithm is an algorithm used to interpret service entry (e.g. service name, service input/output) from the service repository. The second algorithm is an algorithm used to extract words from a list of service goals. The third algorithm is an algorithm used to calculate a semantic distance between ontology nodes. The fourth algorithm is an algorithm used to determine a connection between services (i.e. whether an output of one service could be used as an input of another service). The fifth algorithm is an algorithm used to organize a list of services. The sixth algorithm is an algorithm used to determine an order of service composition based on a result given by an intelligence NLP-tool.

These algorithms (except the sixth algorithm) are based on divide and conquer (i.e. recursion) techniques. The recursion technique has been employed due to its simplicity. However, one should note on a main disadvantage of this technique. That is the algorithm requires large amount of memory [47].

We also present APIs that we have used to develop IA at the end of this appendix. Among other things, IA has been developed by utilizing five APIs, namely, SWT designer API [36], Jena API [20], OpenNLP API [37], KMF API [6] and DOM API [45]. The first three are API developed in an open source project while the fourth is API developed in SPICE project. The DOM API is a part of W3C standard [45].

A.1 Algorithm: Service Repository Interpretation

IA invokes a method call, *parse()*, defined in *ServiceRepository.java* to load all service entries in the repository to its data structure. As mentioned in section 3.4.2, a service entry composes of six elements. IA extracts each element and put it into its associated field. For example, after extraction of service entry (Figure A-1), *MessagingServer*, each field of data structures in Figure 71 keeps the following element:

The *component* field stores a text, "MessagingServer". The *operation* field stores a text, "SmsSender". The *goals* field stores a text, "SendSms". The *input* field stores a text, "Text,SmsAddress. The *output* field stores a text, "N/A". Lastly, the *nf* field stores a text, "5".

"MessagingServer"	"SmsSender"	"SendSms"	"Text,SmsAddress"	"N/A"	2
-------------------	-------------	-----------	-------------------	-------	---

Figure A-1: An example of Service entry in service repository

In the subsequence process, IA will refer to these fields. For example, IA refers to the *input* and *output* fields when it organizes a list of service candidates. However, IA cannot utilize the information in *input* and *output* field directly. Specifically, IA needs a capability to interpret the service *input* and *output* field.

Giving a set of fields in preceding paragraph, IA interprets that MessagingServer requires two input parameters, namely, Text and SmsAddress. Moreover, it also interprets that MessagingServer produces no output. To interpret *output* field, a conditional expression is established to associate a keyword “N/A” with an empty output. A problematic part is an interpretation of the *input* field. To interpret *input* field, IA needs to extract two input parameters, *Text* and *SmsAddress*, from the string “Text,SmsAddress”.

The IA extraction algorithm is:

- 1) Read the entire String (i.e. “Text,SmsAddress”) into temporary String, x
- 2) Get the two substrings of x. The first substring is a set of characters starting at the beginning position of x (i.e. index 0) and ending at a character right before comma. The second substring is a set of characters starting at a character right after comma and ending at the end position of x (i.e. index x.length – 1). If the string x does not contain comma, the first substring is entire string, x. The second string is empty string.
- 3) Put the first substring into an array of input parameters
- 4) Use the second substring as the entire String. Repeat the extraction process recursively until the second substring is empty

By employing the extraction algorithm, the String, “Text,SmsAddress” is interpreted as a list of two input parameters, namely, Text and SmsAddress. Indeed, if a service entry were to have produced more than one output, the same algorithm could be used to interpret the list of service output parameters.

A.2 Algorithm: Word Extraction

A service goal could be composed of more than one word. For example, a service goal, *SendSms*, composes of two words, namely, *Send* and *Sms*. Each of these words is accounted for a maximum weight of the service goal (section 3.5.4). In other words, a maximum weight of service goal is equal to a summation of a weight of Send and a weight of Sms. Therefore, a maximum weight for a service goal can be calculated only when a list of words (e.g. Send and Sms) is known. IA utilizes the subsequence algorithm to extract a list of words from a service goal:

- 1) Read the entire String (i.e. “SendSms”) into temporary String, x
- 2) Get two substrings of x. The first substring is a set of characters starting at the beginning position of x (i.e. index 0) and ending at a character right before an occurrence of a capital letter after index 0 (i.e. ‘d’). The second substring is a set of characters starting

- at the capital letter and ending at the end of string. If there is no capital letter after index 0 exists, the substring is the entire string, x. The second string is empty string.
- 3) Put the first substring into a list of words
 - 4) Use the second substring as the entire string. Repeat the extraction process recursively until the second substring is empty

Essentially, the algorithm is based on an assumption that a word starts with capital letter and follows by small letters. This assumption puts a restriction on definitions of service goal in both service repository and goal ontology. For example, a goal of a service, *MessagingServer*, is defined as *SendSms* rather than *sendsms*. This assumption also simplifies a process of keyword matching. Suppose that an end user requests to “get the weather at Paris and send it to the mobile of John.Doe” and we have a service, *GBSCTHolidayDates* that has a goal, “*GetHolidayDatesInScotland*”, in service repository. Only the word “get” is matched with the word “Get” in the service goal. The word “and” in the user request is not matched with the word, “and”, in the service goal. This is because “and” is not a word in the service goal but, rather, “and” is a part of a word Scotland (that starts with a capital letter). To be more programmatically, IA converts each word in a request to its desired format (i.e. a word starts with capital letter and follows by small letters). Then, IA matches the converted word with service goals. The matching process is case-sensitive. Therefore, the converted word, “And”, could not be matched to “and” of Scotland.

A.3 Algorithm: Service Distance

IA filters out services that have goal weight less than certain threshold. The remaining services are put into a list of consolidated services (section 3.5.4). The goal weight measures a likeliness that the request refers to a service goal. The goal weight is calculated by adding all keywords in the request that correspond to the service goal. Suppose that an end user requests to “get weather report in Paris and send it to my email” and we have a service, *WeatherForecast* that has a goal, “*WeatherForecast*”, in service repository.

Apparently, a word that corresponds to a goal “*WeatherForecast*” is “weather”. Therefore, a goal weight of “*WeatherForecast*” is equal to a weight of “weather”. A maximum weight for “*WeatherForecast*” is a summation of a weight of “weather” and a weight of “forecast”. In section 3.5.4, we have discussed how the weights are calculated. To compute a weight, IA needs to deduce semantic distances between ontology classes in goal ontology. IA utilizes the subsequence algorithm to calculate semantic distances between two ontology classes:

- 2) Start from an ontology class x, we are searching a way to ontology class y. Set a semantic distance, d, to zero. Initialize a list of ontology class. On initialization, the list does not contain any ontology classes
- 3) Check if ontology class x is, in fact, an ontology class y. If this condition satisfies, return a semantic distance, d. Otherwise, goes to step 3
- 4) Find semantic distances of superclasses of x. A semantic distance of x-to-y via superclass is the least semantic distance plus one. The least semantic distance is a semantic distance of a superclass of x-to-y that is less than the semantic distances of other superclasses of x-to-y. To find a semantic distance of superclass of x-to-y, we add a superclass of x to the list, l. Then, we recursively follow a step of finding a semantic distance of x-to-y with a superclass of x if the superclass is not in the list. Otherwise, the superclass forms a loop.

The loop does not lead to the least semantic distance and, therefore, we can rule out this recursive thread. To clarify on this point, we give an example of a loop. Suppose that we want to find a semantic distance from A to B, the loop can be A->C->D->C->B. Clearly, the thread A->C->B does exist therefore we can ignore the recursive branch that leads to the loop

- 5) Find semantic distances of subclasses of x-to-y. A semantic distance of x-to-y via subclass is the least semantic distance plus one. The least semantic distance is a semantic distance of a subclass of x-to-y that is less than the semantic distances of other subclasses of x-to-y. To find a semantic distance of subclass of x-to-y, we add a subclass of x to the list, l. Then, we recursively follow a step of finding a semantic distance of x-to-y with a subclass of x if the subclass is not in the list. Otherwise, the subclass forms a loop. The loop does not lead to the least semantic distance and, therefore, we can rule out this recursive thread. To clarify on this point, we give an example of a loop. Suppose that we want to find a semantic distance from A to B, the loop can be A->C->D->C->B. Clearly, the thread A->C->B does exist therefore we can ignore the recursive branch that leads to the loop
- 6) A semantic distance of x-to-y is a semantic distance of x-to-y via superclass if it is less than a semantic distance of x-to-y via subclass. Otherwise, a semantic distance of x-to-y is a semantic distance of x-to-y via subclass

The algorithm is based on an assumption that the ontology is well structured. In other words, it is based on an assumption that there always be at least one path leads from one ontology class to another ontology class. However, the assumption does not put a restriction on the ontology hierarchy. For example, one ontology class could have more than one superclass or subclass or it could have only one superclass or subclass. As IA traverses the ontology using only superclass and subclass relation, it is flexible to support multiple definitions of ontology that are based on different hierarchy.

A.4 Algorithm: Service Connections

In order to compose for a service, we have to determine whether services could be associated with each other. Two services could be combined if an output of one service could be used as an input of another service. In other words, two services could be combined if they are compatible. The subsequence algorithm is used to determine service compatibility:

- 1) Start from loading a list of input ontology classes, l1, of one service and a list of output ontology classes, l2, of another service
- 2) Check whether one of the ontology classes in the list, l1, has a semantic similarity with one of the ontology classes in the list, l2. This is done by recursively call a method to check a compatibility of an input of one service and an output of another service. If the condition satisfies, two services are compatible

The subsequence algorithm is used to determine semantic similarity:

- 1) Start from loading an input ontology class, C1, of one service, S1, and an output ontology class, C2, of another service, S2. Initialize a semantic distance, d, between two services to be zero

- 2) Check whether the input ontology class $C1$ is equal to output ontology class $C2$. If the condition satisfies, the service, $S1$, is semantically similar to the service, $S2$ with a semantic distance, d . Otherwise, go to step 3.
- 3) Load a list of superclasses, $SL1$, of the ontology class, $C1$. If the list $SL1$ could not be loaded (i.e. the ontology class, $C1$, does not have a parent), go to step 4. Otherwise, increment the semantic distance, d , by one. Then, set ontology class, $C1$ to be one ontology class, $P1$, of the list $SL1$ at a time and recursively follow step 2 through 3 until the recursion terminates. The recursion is terminated if one of the following conditions satisfied:
 - Service, $S1$, is found to be similar to service, $S2$
 - The list of superclass, $SL1$, could not be loaded
- 4) Start from loading an output ontology class, $C2$, of one service, $S2$, and an input ontology class, $C1$, of another service, $S1$. Initialize a semantic distance, d , between two services to be zero
- 5) Check whether the output ontology class $C2$ is equal to input ontology class $C1$. If the condition satisfies, the service, $S1$, is semantically similar to the service, $S2$ with a semantic distance, d . Otherwise, go to step 6
- 6) Load a list of superclasses, $SL1$, of the ontology class, $C2$. If the list $SL1$ could not be loaded (i.e. the ontology class, $C2$, does not have a parent), go to step 7. Otherwise, increment the semantic distance, d , by one. Then, set ontology class, $C2$ to be one ontology class, $P1$, of the list $SL1$ at a time and recursively follow step 5 through 6 until the recursion terminates. The recursion is terminated if one of the following conditions satisfied:
 - Service, $S1$, is found to be similar to service, $S2$
 - The list of superclass, $SL1$, could not be loaded
- 7) Start from loading an input ontology class, $C1$, of one service, $S1$, and an output ontology class, $C2$, of another service, $S2$. Initialize a semantic distance, d , between two services to be zero
- 8) Check whether the input ontology class $C1$ is equal to output ontology class $C2$. If the condition satisfies, the service, $S1$, is semantically similar to the service, $S2$ with a semantic distance, d . Otherwise, go to step 9
- 9) Load a list of superclasses, $SL1$, of the ontology class, $C1$. Load a list of superclasses, $SL2$, of the ontology class, $C2$. If the list $SL1$ or $SL2$ could not be loaded (i.e. the ontology class, $C1$, does not have a parent or the ontology class, $C2$, does not have a parent), go to step 10. Otherwise, increment the semantic distance, d , by one. Then, set ontology class, $C1$ to be one ontology class, $P1$, of the list $SL1$ at a time. Also, set ontology class, $C2$, to be one ontology class, $P2$, of the list $SL2$ at a time. This way ontology class, $P1$, is used as a pivot. We search for an ontology class, X , in the list $SL2$ that is semantically similar to $P1$. Recursively, follow step 8 through 9 until the recursion terminates. The recursion is terminated if one of the following conditions satisfied:

- Service, S1, is found to be similar to service, S2
 - The list of superclass, SL1 or SL2 could not be loaded
- 10) Start from loading an input ontology class, C1, of one service, S1, and an output ontology class, C2, of another service, S2. Initialize a semantic distance, d, between two services to be zero
 - 11) Check whether the input ontology class C1 is equal to output ontology class C2. If the condition satisfies, the service, S1, is semantically similar to the service, S2 with a semantic distance, d. Otherwise, go to step 12
 - 12) Load a list of superclasses, SL1, of the ontology class, C1. Load a list of superclasses, SL2, of the ontology class, C2. If the list SL1 or SL2 could not be loaded (i.e. the ontology class, C1, does not have a parent or the ontology class, C2, does not have a parent), go to step 13. Otherwise, increment the semantic distance, d, by one. Then, set ontology class, C2 to be one ontology class, P2, of the list SL2 at a time. Also, set ontology class, C1, to be one ontology class, P1, of the list SL1 at a time. This way ontology class, P2, is used as a pivot. We search for an ontology class, X, in the list SL1 that is semantically similar to P2. Recursively, follow step 11 through 12 until the recursion terminates. The recursion is terminated if one of the following conditions satisfied:
 - Service, S2, is found to be similar to service, S1
 - The list of superclass, SL2 or SL1 could not be loaded

The algorithm returns either: 1) the two services; S1 and S2 have semantically similarity with a semantic distance, d or 2) the two services; S1 and S2 do not have semantically similarity. If the latter is the case, the two services could not be composed. It is worth mentioning that we ignore the case where two services could be run in parallel. In other words, we ignore the case where two services could be composed even when they do not use an output of the others. Therefore, in our work, we focus on a sequential composition (when one service use output of another service as its input).

A.5 Algorithm: Service Composition

In section 3.5.5, we have discussed how a list of consolidates services are organized before service composition. In this section, we are going to discuss about an algorithm to organize the list of services in more technical details. The subsequence algorithm is used to organize the list of services:

- 1) Start from a list of consolidated services, l. Initialize a list of service composition, ll. On initialization, the list of service composition does not contain any composition entries
- 2) Pick a service, S, which had never been picked before, from a list of consolidated services. Initialize a service composition, C. The service composition, C, is a list of service components. On initialization, the service composition does not contain any service components. Put the service, S, into the list, C. At this point, the service, S, is at a tail of the list, C.

- 3) Find a list of services, l_2 , which could provide its outputs as an input of a service, S_1 , from a list of remaining consolidated services. The service, S_1 , is a service at a tail of the list, C . If the list, l_2 , is empty, skip to step 4. Otherwise, for each service, S_2 , in the list l_2 , put S_2 into the service composition, C . If the service composition, C , had already been inserted with one of services in the list, l_2 , copy the service composition, C , into another service composition, C_2 . Then, insert the service, S_2 , into the list, C_2 . In the end, a number of service compositions created on this round will be equal to a size of the list, l_2
- 4) Put all service compositions created in step 3, to a list of service compositions, l_1 . For each of the service composition, C , in the list, l_1 , recursively, follow step 3 through 4 until step 3 is completed on the condition that the list, l_2 is empty
- 5) In the end, all service compositions are in the list, l_1

As implied in step 3 of the algorithm, number of service compositions is equal to a number of consolidated services multiplies by a number of composition branches for each service. Numbers of composition branches for a service (S) depend on an appropriateness of other services to provide an output as an input of the service, S . If a number of consolidated services are 4 and every service can be linked to each other, number of service compositions will be $4 \times 3 \times 2 \times 1$ or 24 compositions. Suppose that we have a list of consolidated services (A, B, C, D). Starting from service A , the first composition branch (i.e. possible composition of two service) is $A \rightarrow B, A \rightarrow C, A \rightarrow D$. The second composition branch is $A \rightarrow B \rightarrow C, A \rightarrow B \rightarrow D, A \rightarrow C \rightarrow B, A \rightarrow C \rightarrow D, A \rightarrow D \rightarrow B, A \rightarrow D \rightarrow C$. The third composition branch (where the service composition is completed) is $A \rightarrow B \rightarrow C \rightarrow D, A \rightarrow B \rightarrow D \rightarrow C, A \rightarrow C \rightarrow B \rightarrow D, A \rightarrow C \rightarrow D \rightarrow B, A \rightarrow D \rightarrow B \rightarrow C, A \rightarrow D \rightarrow C \rightarrow B$. Therefore, starting from service A , we have got six service compositions possible. Since we can also start from service B, C and D , the total number of service compositions is 6×4 or 24 compositions as expected.

A.6 Algorithm: Sequence of Event

An outcome, obtained by using keyword extraction technique, is not specific enough. An intelligence NLP-tool is used to provide additional information so that the redundancy compositions are given a low weight (when compare to the more precise compositions).

Specifically, IA analyzes an XML generated by NLP-tool in order to get more information on the user request (and hence, service compositions). The XML is a result of conducting a natural language processing (using semantic analysis technique in section 2.1.3) on the user request. The XML contains information (e.g. a set of predicate, argument list and general quantifier) such that a relationship between services can be deduced.

The relationship determines which services should be invoked before the other services. The relationship is inferred from a sequence of events. IA uses keywords as indicators of events. For example, if user requests to “find traffic information in Paris and send them by email”, IA could deduce from the XML that a service that has goal, *GetTrafficCondition*, is invoked before another service that has goal, *SendEmail*. This is because the first event (that is associated with the word, “traffic”) occurs after the second event (that is associated with the word, “send”). Therefore, if we could determine an order of words, we could deduce a sequence of events. It is worth mentioning that the order of words does not depend on a location of the words in the sentence. Rather, it depends on a meaning of conjunction and preposition that links those

words together. The subsequence algorithm is used to determine an order of two words (i.e. two strings) from the XML file:

- 1) Load two Strings, x and y . We are going to determine whether the String, x , happens before the String, y
- 2) Get a list of arguments, l that has a predicate name equals to y . Recall from section 3.5.8 that IA loads predicates and their argument lists from XML file to its internal memory
- 3) If x exists in the list, l , we conclude that x happens before y and terminate the algorithm. Otherwise, go to step 4
- 4) Get a list of arguments, l_1 that has a predicate name equals to the preposition, “by”. Recall from section 3.5.8 that IA loads predicates and their argument lists from XML file to its internal memory
- 5) If x happens before y in the list, l_1 , we conclude that x happens before y and terminate the algorithm. Otherwise, go to step 6
- 6) Get a list of arguments, l_2 that has a predicate name equals to the preposition, “in”. Recall from section 3.5.8 that IA loads predicates and their argument lists from XML file to its internal memory
- 7) If x happens before y in the list, l_2 , we conclude that x happens before y and terminate the algorithm. Otherwise, go to step 8
- 8) Get a list of arguments, l_3 that has a predicate name equals to the preposition, “and”. Recall from section 3.5.8 that IA loads predicates and their argument lists from XML file to its internal memory
- 9) If x happens before y in the list, l_3 , we conclude that x happens before y and terminate the algorithm. Otherwise, we conclude that x does not happen before y and terminate the program

In fact, the two strings x and y are the two words in the input sentence (i.e. user request). NLP-tool analyzes the input sentence and associates every words in the sentence in a form of predicate and argument list. Moreover, the two strings x and y correspond to service goals in service repository. If string x happens before string y , we conclude that service that has a goal corresponding to string x is invoked before service that has a goal corresponding to string y .

A.7 APIs

IA has been developed by utilizing fifth APIs, namely, SWT designer API [36], Jena API [20], OpenNLP API [37], KMF API [6] and DOM API [45]. The first three are APIs developed in open source projects while the fourth is API developed in SPICE project. In this section, we will elaborate on a core part of the APIs that plays an important role in our IA prototype development.

A.7.1 SWT Designer

SWT Designer is used to develop an interface of IA. The SWT Designer is preferred over the other APIs such as Java swing [48] because it is an Eclipse plug-in. Since the development of IA has been done on Eclipse platform, SWT Designer is an ideal option.

The interface has been designed through a **Graphical User Interface (GUI)** of SWT Designer. After elements (e.g. radio box, button etc.) are drawn into the interface panel, a skeleton code is created, correspondingly. Eventually, the real content of event handlers (i.e. methods that are invoked after some certain events happen (e.g. a click on a button)) are added through a skeleton code as shown in Figure A-2 (taken from source file, UI.java). The *widgetSelected()* is an event handler that will be invoked after a submission button (i.e. *requestButton*) is pressed.

```
requestButton.addSelectionListener(new SelectionAdapter() {
    public void widgetSelected(final SelectionEvent arg0) {
        enableRequestButtons();
        try{
            outputText.setText("");
            serviceText.setText("");
            detailText.setText("");
            //outputText.setText(parser.parses(requestText.getText()));

            detailText.setText(parser.parses(requestText.getText()));
            Integer x = Test.getInt();
            outputText.setText(x.toString() + " compositions have been exported
... [result folder = spatel/]");

            serviceText.setText(x.toString());
        }catch(Exception e){
            System.out.println("Parser failed!");
            e.printStackTrace();
        }
    }
});
```

Figure A-2: An event handler of UI

A.7.2 Jena API

IA employs Jena API to read and analyze ontologies. As discussed in 3.4, there are three main ontologies: Goal ontology, Input/Output ontology and Synonym ontology. To access concepts in the ontologies, IA traverses on structure of ontologies and searches for the concepts. Specifically, IA employs the following methods of Jena API to operate on the ontologies (these are defined in source file, OntologyReader.java):

- Initially, IA loads the ontology (e.g. Goal ontology, Input/Output ontology) into a java object, OntModel via a method *model.read()*. This is shown in Figure A-3
- After the model has been loaded, the concepts defined in ontology could be accessed via methods: *model.getOntClass(uri)*, *child.listSuperClasses()*, *child.getSuperClass()*, *parent.listSubClasses()*, *model.listClasses()* and, *ontclass.listEquivalentClasses()*. *model.getOntClass(uri)* retrieves ontology concept that is identified by URI. The method returns an ontology concept in a form of a Java object, OntClass. *child.listSuperClasses()* gets all superclasses of child (child is an OntClass). Recall that concepts in ontology are organized in a similar way as those of **Object-oriented pro-**

gramming language (OOP) (i.e. it has superclass, subclass relationship). Unlike OOP such as Java, a class (i.e. concept) can be inherited from multiple parents (i.e. concepts). If an ontology class (i.e. concept) has only one parent, the parent (the more general concept) can be obtained through method, *child.getSuperClass()*. *parent.listSubClasses()* gets all subclasses of parent (parent is an *OntClass*). *model.listClasses()* gets all concepts (that are of type *OntClass*) defined in an ontology. While concepts in Goal ontology and Input/Output ontology are accessed through superclass-subclass relationships, concepts in Synonym ontology are connected through equivalence relationship (section 2.2). As a result, concepts in Synonym ontology are accessed through method *ontclass.listEquivalentClasses()*. For example, to find words those are synonym of “find”. Initially, we load Synonym ontology into an ontology model, *OntModel*. Then, we get a concept (that has type *OntClass*) of “find” through a method, *model.getOntClass(uri)* (*uri* is an identification of find which is *<uri of ontology>#find*). From the *OntClass* of “find”, we get all *OntClass* that is equivalence (i.e. synonym of) to the *OntClass* through a method, *ontclass.listEquivalentClasses()*

- IA uses the following judgment to compare two concepts. Two concepts are equal if they refer to the same URI. Otherwise, they are not equal. This judgment is used, for instance, when IA finds a semantic distance between two concepts. Start from the first concept, IA traverses through a structure of ontology (via method, *listSuperClasses()*, *listSubClasses()*) until it finds the second concept that is equal to the first concept

Another use of Jena API is to analyze user contexts retrieved from the Context Source. As discussed in section 3.5.12, the user contexts are published in RDF. IA employs the following methods of Jena API to operate on the context (these are defined in source file, *RdfReader.java*):

- Initially, IA loads the context into a java object, *Model* via a method *model.read()*. This is shown in Figure A-4. *Model* is more general than *OntModel*. In fact, it is general enough to hold RDF model
- Similar to a concept in ontology, an RDF property is accessed through a method, *model.getProperty(uri)* (*uri* is an identification of property)
- To find an RDF statement (section 2.2.1), the method, *model.listStatements(selector)* is used. The *selector* is used to hold a triple described in section 2.2.1. The triple comprises of *<subject,predicate,object>*. Giving a subject and predicate, all RDF statements in the model, that have triple contains the subject and predicate (despite their object), are returned. To be more specific, subject is a resource that contains a predicate which is property. The resource is obtained from a method, *model.listObjectsOfProperty(property)*
- Once the RDF statement (that corresponds to property) is obtained, the relevant contexts will be derived. For example, IA searches for RDF statement that corresponds to a property, *isLocatedIn*. The object of this RDF statement is a location of the user. Based on this location, certain context rules are enforced

```

model = ModelFactory.createOntologyModel();
ontologyURL = (new File(modelFile)).toURL().toString();

```

```
model.read(ontologyURL);
```

Figure A-3: To load ontology model (OWL)

```
model = ModelFactory.createDefaultModel();
byte currentXMLBytes[] = contexts.getBytes();
ByteArrayInputStream byteArrayInputStream = new ByteArrayInputStream(
Stream(currentXMLBytes);
model.read(byteArrayInputStream, "");
model.write(System.out);
```

Figure A-4: To load ontology model (RDF)

A.7.3 OpenNLP API

IA employs OpenNLP API to parse user requests (i.e. to assign grammatical element (noun, verb, etc.) to each word in the user requests). Parsing user requests is an initial step toward service discovery (based on keyword searching approach). IA employs the following methods of OpenNLP API to parse the user request (these are defined in the method *parses()*, *getLexicon()* of the source file, UIParser.java):

- Initially, IA employs a method, *tokenize(sentence)* of class, Tokenizer, to break apart the sentence (a word in a sentence is a token. A sentence is a user request). This is shown in Figure A-5
- IA converts each token to a form that is compatible with OpenNLP API. For example, “(“ is converted to “-LRB-”.
- From a list of tokens, IA constructs a sentence as shown in Figure A-6
- IA creates a root node of the parse tree by constructing a Parse object on the entire sentence (the entire sentence is given as parameter of Parse object). This process is shown in Figure A-7
- The other nodes of the parse tree is created for each token and added to a list of nodes as shown in Figure A-8
- A parse tree is constructed from a list of nodes by calling a method *parse(list_of_nodes,number)* of ParserME, object. This is shown in Figure A-9. ParserME is a specific type of Parser objects. ParserME is constructed from a method *Parser.getParser()*
- To access a node in the parse tree, IA calls methods, *Parse.getChildren()*, *Parse.getType()*. Start from a root node, IA uses method *getChildren()* recursively to get all other nodes in the tree. IA also uses method *getType()* to check whether the nodes are one of the IA desired types (i.e. noun, verb)

```
String[] tokens = tokenizer.tokenize(sent);
```

Figure A-5: To break apart the sentence

```
StringBuffer sb = new StringBuffer();
List<String> tokenList = new ArrayList<String>();
```

```
for (int j = 0; j < tokens.length; j++)
{
    String tok = convertToken(tokens[j]);
    tokenList.add(tok);
    sb.append(tok).append(" ");
}
String text = sb.substring(0, sb.length() - 1).toString();
```

Figure A-6: To construct a sentence from a list of tokens

```
// the parent parse instance spans the entire sentence
Parse p = new Parse(text, new Span(0, text.length()), "INC", 1, null);
```

Figure A-7: To construct root node of parse tree

```
// create a parse object for each token and add it to the parent
int start = 0;
for (Iterator ti = tokenList.iterator(); ti.hasNext();)
{
    String tok = (String) ti.next();
    p.insert(new Parse(text, new Span(start, start + tok.length()),
        ParserME.TOK_NODE, 0));
    start += tok.length() + 1;
}
```

Figure A-8: To construct child nodes of parse tree

```
// fetch multiple possible parse trees
int numParses=1;
Parse[] parses = parser.parse(p,numParses);
```

Figure A-9: To construct a parse tree

A.7.4 KMF API

IA employs KMF API to perform context handling. There are two types of KMF API: the publish API and subscribe API. In this section, we will focus on subscribe API in which the IA deploys to retrieve certain user contexts published by some context sources in the KMF architecture. When the end user enables context handling feature of IA, the context subscription process begins. IA employs the following methods of KMF API to subscribe for the user contexts (these are defined in the source file, *ContextSink.java*):

- Initially, IA creates listener for certain user context (e.g. location context). This is shown in Figure A-10
- Then, IA exports the listener as an interface that can be invoked through **Remote Procedure Call (RPC)** [49]. This is done via *ExportManager*, object. *ExportManager* is a utility class. It is a part of KMF API. The listener will be called by Knowledge Broker as soon as the certain user context (in this case location context) occurs. This is shown in Figure A-11
- Next, IA creates an authentication helper which handles authentication. The authentication helper is defined as Java object, *AAHelper*. It takes a user name and password as its

parameter. The user name and password has to be corresponding to the one in credential files. The credential files [35] are local files defining a permission of the certain user (these files contained user name and password. Since, in this thesis, we do not concern about KMF security. We shall not elaborate on the credential files). In essence, the user needs to have permission to access location context. This is shown in Figure A-11

- Then, IA creates a subscription helper. The subscription helper is defined as Java object, *ContextSubscriptionHelper*. The *ContextSubscriptionHelper* is a utility class that is used to handle publish-subscribe request (recall that there are two types of request, namely, request-response and publish-subscribe). Through the subscription helper, IA registers its interest to subscribe for certain user context (e.g. location context). Eventually, IA starts subscribing with a rescan interval equals to 60 seconds (i.e. IA searches for a new context source (of a certain type of context) every 60 seconds). The subscribe method returns a subscription key. The subscription key is an identification that is used to identify a particular subscription. This is shown in Figure A-11
- If the end user disable context-handling feature, IA will unsubscribe user contexts through a method *removeContextSubscription(key)* of *ContextSubscriptionHelper*. *key* is a subscription key mentioned in previous paragraph. This is shown in Figure A-12

In fact, when IA subscribes for user context, context-handling thread will be created. This thread is automatically created through KMF API when the subscribe method is called. This is a reason why IA can operate on user request and handle user context in parallel. The user request is handled by IA main thread while the context is handled by context-handling thread.

Technically, there are two context-handling threads. The first context-handling thread (mentioned in preceding paragraph) is automatically created through KMF API. The second context-handling thread is manually created by IA. This approach improves performance of IA, especially in multiprocessor machine. While the first thread focuses on context-updating process, the second thread can concentrate on context-handling process.

If the certain user context occurs, the first context-handling thread will set up a context variable (i.e. the context variable keeps user context in RDF format). The context variable will be checked by the second context-handling thread. If the context variable is updated with a new context, the second context-handling thread will enforce context-handling rule, accordingly (section 3.5.13)

```
// create listener for location context
locationListener = new LocationListener();
```

Figure A-10: To construct context listener

```
// create an authentication helper which handles authentication, refreshing tokens, etc.
AAHelper aaHelper = new AAHelper(new NamePasswordCredentials("Admin", "AdminSecretS"), null);
// create a subscription helper
subscriptionHelper = new ContextSubscriptionHelper(aaHelper);
// we're interested in location updates
key = subscriptionHelper.addContextSubscription(UserLocation.TYPE.getURI(), locationListener, true);
```

```
// start subscribing, rescan for new sources every 60 seconds
subscriptionHelper.start(60000);
```

Figure A-11: IA Subscription process

```
if(key!=null)subscriptionHelper.removeContextSubscription(key);
```

Figure A-12: IA Un-subscription process

A.7.5 DOM API

IA employs DOM API [45] to read and analyze an MRS created by NLP-tool. As discussed in section 3.4.9, the MRS is constructed in XML format. Thus, the MRS forms a tree structure in XML. This XML-tree is parsed and loaded into a Java Object, DOM. DOM is a standard Java object defined by W3C. DOM API is used to operate on XML document. Specifically, IA employs the following methods of DOM API to process the MRS created in XML format (these are defined in source file, `xmlParser.java`):

- Initially, IA parses the XML via method `DomParser.parse(DataSource(file))`. `DomParser` is a utility class that is used to parse the XML file. The XML file is identified as an `DataSource` object (`DataSource` is a Java object defined in `java.io` package). This is shown in Figure A-13
- After the XML structure has been parsed, it is loaded into a java object, `Document`. `Document` is a utility class that is used to keep an XML tree. `Document` provides several useful methods to query and extract nodes (i.e. elements) from the XML tree. XML node comprises of a list of tag names, and a list of node values. This is shown in Figure A-13
- To get XML nodes [45] that corresponds to specified XML tag, IA employs a method, `getElementsByTagName(tag)` of `Document`. This is shown in Figure A-13
- The method, `getElementsByTagName(tag)` returns a list of nodes (i.e. it returns an object of type, `NodeList`). To get a node (i.e. of Java type `Node`) from the list, IA employs a method, `item(index)` of `NodeList`. `index` indicates a position of the node in the list. This is shown in Figure A-13
- To get child nodes of the node, `x`, IA employs a method, `getChildNodes()` of `Node`. This method returns a list of child nodes of `x`. This is shown in Figure A-13
- To get node value (i.e. the string between XML tag), IA employs a method, `getNodeValue()` of `Node`. This is shown in Figure A-13
- XML node comprises of other XML nodes. For example, in Figure A-14, the document node comprises of two XML nodes (that correspond to tag `<spread>`). If IA employs a method, `getElementsByTagName(spread)` on document node, these two node will be returned. If IA requires only the first node to return, it will need to employ a method, `getElementsByTagName(ep)` on the document node. This method will return two nodes that correspond to tag `<ep>`. Then, IA employs a method, `getElementsByTagName(spread)` on the first node (that corresponds to `<ep>`). As a result, only the first node corresponding to `<spread>` will be returned. Technically, `getElementsByTag-`

Name(tag) is not a member function of *Node*. IA needs to cast *Node* to *Element* (a more specific type of node) in order to traverse inside the XML node (i.e. invoke the method *getElementsByTagName(tag)*)

Essentially, IA traverses on XML structure and constructs a set of predicates (i.e. handles). The predicates are constructed as follows:

IA gets a list of XML nodes, *l*, corresponding to the tag, `<ep>`

For each XML node, *n*, (from the list, *l*) IA gets a node, *n'*, corresponding to the tag, `<spred>` (there is only one tag `<spred>` for each `<ep>`). The value of a node, *n'*, is obtained from getting a child node, *c*, of *n'* and get the value of *c*. The value of a node *n'* is used as a **predicate name** providing that it is not one of the types, general quantifier (e.g. the one that ends with `_q_rel`).

For each XML node, *n*, IA gets a node, *n''*, corresponding to the tag, `<fvpair>` (there is only one tag `<fvpair>` for each `<ep>`). For a node, *n''*, IA gets a list of nodes, *l'''*, corresponding to the tag, `<rargname>`, and a list of nodes, *l''''*, corresponding to the tag, `<var>`. The node, *n''''* (from a list of nodes, *l''''*), has an attribute “sort” and attribute “vid”. These attributes combined to form a variable.

Depending on the value of node, *n''''*, the variable is either an identification of the **predicate name** (if the value of node, *n''''*, equals to “ARG0”) or one of identifications of the predicate’s arguments (if the value of node, *n''''* (i.e. argument type), equals to “ARG1”, “ARG2” etc). Normally, after the set of arguments are obtained, IA constructs a predicate that comprises of predicate name, argument type and the argument values (i.e. variables).

However, if the value of node, *n'*, is referred to co-referential relationship (i.e. `coreferential_rel`), the relationship between variables (i.e. two variables refer to the same predicate) are added to co-referential table. Moreover, if the value of node, *n'*, is referred to named relationship (i.e. an argument of predicate (i.e. variable) indirectly refers to another variable, *v2*, that is used to indicate a constant (e.g. “Paris”)), IA searches for the variable, *v2*, by starting from a node, *n''''*, of the list, *l''''*, until IA finds a node *n''''* that has an argument type equals to “CARG”. The argument value of this node is used as a variable. Then, the predicate is constructed as before.

```
p.parse(new InputSource(in));
Document doc = p.getDocument();
NodeList nl = doc.getElementsByTagName("ep");

for(int i=0;i<nl.getLength();i++){
    Element e = (Element)nl.item(i);
    NodeList el01 = e.getElementsByTagName("spred");
    String engPred = "";
    String nameTag = "";
    String myargument = "";

    // ignore general quantifier
    if(el01.item(0) != null &&
        (el01.item(0).getChildNodes().item(0).getNodeValue().endsWith("_q_rel")
        ||
        el01.item(0).getChildNodes().item(0).getNodeValue().endsWith("pron_rel"))) continue;
```

Figure A-13: An example of DOM API

```
<ep><spred>_example1_<\spred><\ep>
<ep><spred>_example2_<\spred><\ep>
```

Figure A-14: An example of XML nodes

Appendix B

Problems during implementation

As a scale of the thesis work is large, there are some difficulties found in the project. The most difficulties were found during the design and implementation phrase of IA. The plan has been deviated three times and they lead to modifications of project schedule. An original plan of the project will be sketched out in this chapter while the root cause of the problems and the solution to the problems are discussed.

B.1 The First Modification on Project Schedule

In the end of February, we have found out that the phrase one (i.e. to collect all user requirements) of the project has been extended from the middle of February to the first week of March. After a technical meeting with natural language processing team, we can conclude the IA features and design. The three user scenarios (defined in chapter 5) have been obtained and reflected in the project plan (i.e. the draft user scenarios have been amended). The problems that lead to the first plan modification are:

1. The Natural Language Processing technology itself is quite a new field for the project group and therefore we rely on collaboration with external party. This makes it sometimes, difficult to have a full control on a time frame
2. Although the requirement was clear from the beginning, the way to achieve it is not clear. It is figured out later than expected

The modification on the plan did not defect the time schedule because we had conducted the phrase two in parallel (e.g. some general functions, such as a context retrieving feature, had already been implemented as planned).

B.2 The Second Modification on Project Schedule

In the middle of March, we have found out that the phrase 2 is delayed (i.e. according to the original plan, we should have finished it on March, 16th). This makes us unable to follow the original plan. We have noticed that the phrase 2 and 3 of the project are pretty much leading to the same functionalities. Thus, they could be merged. Furthermore, we have also noticed that the phrase 4 and phrase 5 of the project are also leading to the same functionalities. Thus, they could also be merged. We had estimated the time for original plan equally. At this point, we had noticed that they are not equal. For phrase 4 and phrase 5, we have an API available to

Problems during implementation

handle all contexts and thus they require approximately one month to be completed. For phrase 2 and 3, we realized that we had underestimated the plan as we had been thinking to reuse some of the existing technologies but it was clear that the technologies are not suitable for our purpose and we need to invent the new application. To be more specific, the problems that lead to the second plan modification are:

1. The available service repository and ontology files are not sufficient to carry out the useful samples, we need to introduce the new ones
2. Some of the project phrases were divided even though they could be merged providing that they were related to the same functionality of IA
3. We underestimate some of the project phrases while overestimate some of the others
4. Although the requirement was clear from the beginning, the way to achieve it is not clear. It is figured out later than expected
5. The Natural Language Processing technology itself is quite a new field for the project group and therefore we rely on collaboration with external party. This makes it sometimes, difficult to have a full control on a time frame

The solutions to the problems are that we noticed that we had overestimated some of the project phrases; therefore, the underestimated project phrases can be shifted to use the overestimated time slots. This makes the plan still be on schedule.

B.3 The Third Modification on Project Schedule

In the beginning of May, we have found out that the use-case 2 is delayed (i.e. according to the original plan, we should have finished it on March, 31th). This makes us unable to follow the original plan. To be more specific, the problems that lead to the second plan modification are:

1. We had underestimated the complexity when we integrated IA and ACE. The integration was finished in the end of March (i.e. this is later than expected)
2. Following from point 1, the meeting with NLP group had been postponed. Therefore, the agreement (i.e. requirements) on the integration between IA and NLP result had been lately obtained

As for an impact of this delay to the overall project schedule, there is no major impact because our IA is flexible enough to analyze the NLP-result in Norwegian language by deploying dictionary file (i.e. a language was a major concern to integrate between IA and NLP tool) Hence, the integration process will not take long time (at most 1 weeks, in which, most of the time is on the NLP side and therefore, this is equivalence to author time of three or four days).

Moreover, we had also proceeded with the implementation of location awareness features as planned while we were waiting for the meeting (mentioned in 2). However, this implies that we requires three or four days in addition. In order to stick with the deadline (where, all implementation should be finished at the end of May), the author has decided to add daily extra working hours (+1). As a result, the project schedule is still on the line.

Appendix C

IA configuration files

In this appendix, we present configuration files of IA. These files have been discussed in section 3.4.2 - 3.4.9.

C.1 Service Repository (services.csv)

"Component"	"Operation"	"Goals"	"Inputs"	"Outputs"	"NF"
"PhoneAgent"	"PhoneAgent"	"GetUserInformation"	"UserID"	"CellNumber,IMSI"	5
"GSMLocator"	"CellSpotter"	"FindCellIdLocation"	"CellNumber,IMSI"	"CellID"	1
"GetRestaurantByAddress"	"RestaurantAddressBook"	"FindRestaurantByAddress"	"Address"	"RestaurantAddress"	
"GetRestaurantByCoords"	"NearestRestaurantGPS"	"FindRestaurantByLocation"	"ZipCode"	"RestaurantLocation"	6
"GetRestaurantByCoords"	"NearestRestaurantCellId"	"FindRestaurantByLocation"	"ZipCode"	"RestaurantLocation,RestaurantInfoEnglish"	
"MultilanguageRestaurantGuide"	"RestaurantInfoFrench"	"MultilangRestGuide1"	"TranslateToFrench"	"RestaurantAddress"	
"MultilanguageRestaurantGuide"	"RestaurantInfoFrench"	"MultilangRestGuide2"	"TranslateToFrench"	"RestaurantLocation"	
"MessagingServer"	"SmsSender"	"SendSms"	"Text,SmsAddress"	"N/A"	2
"MessagingServer"	"MmsSender"	"SendMms"	"Media_Content,MmsAddress"	"N/A"	3
"MessagingServer"	"SMTPServer"	"SendInformation"	"Media_Content,EmailAddress"	"N/A"	
"GuideMichelin"	"RestaurantInFrance"	"FindRestaurantInFrance"	"CityName"	"RestaurantInfoFrench"	
"AddressResolver"	"ResolveAddress"	"FindAddressOfLocation"	"GpsCoordinates"	"CityName"	
"GoogleTranslate"	"EnglishToFrench"	"TranslateToFrench"	"EnglishText"	"FrenchText"	
"GoogleTranslate"	"EnglishToNorwegian"	"TranslateToNorwegian"	"EnglishText"	"NorwegianText"	
"GoogleTranslate"	"FrenchToEnglish"	"TranslateToEnglish"	"FrenchText"	"EnglishText"	
"GoogleTranslate"	"FrenchToNorwegian"	"TranslateToNorwegian"	"FrenchText"	"NorwegianText"	
"GoogleTranslate"	"NorwegianToEnglish"	"TranslateToEnglish"	"NorwegianText"	"EnglishText"	
"GoogleTranslate"	"NorwegianToFrench"	"TranslateToFrench"	"NorwegianText"	"FrenchText"	
"WeatherForecast"	"WeatherForecast"	"WeatherForecast"	"CityName"	"Temperature"	
"GoogleTranslate"	"EnglishToHungarish"	"TranslateToHungarish"	"EnglishText"	"HungarishText"	
"GoogleTranslate"	"EnglishToSpanish"	"TranslateToSpanish"	"EnglishText"	"SpanishText"	
"GoogleTranslate"	"FrenchToHungarish"	"TranslateToHungarish"	"FrenchText"	"HungarishText"	
"GoogleTranslate"	"FrenchToSpanish"	"TranslateToSpanish"	"FrenchText"	"SpanishText"	
"GoogleTranslate"	"HungarishToEnglish"	"TranslateToEnglish"	"HungarishText"	"EnglishText"	
"GoogleTranslate"	"HungarishToFrench"	"TranslateToFrench"	"HungarishText"	"FrenchText"	
"GoogleTranslate"	"HungarishToNorwegian"	"TranslateToNorwegian"	"HungarishText"	"NorwegianText"	
"GoogleTranslate"	"HungarishToSpanish"	"TranslateToSpanish"	"HungarishText"	"SpanishText"	

IA configuration files

"GoogleTranslate"	"NorwegianToHungarish"	"TranslateToHungarish"	"NorwegianText"	"HungarishText"
"GoogleTranslate"	"NorwegianToSpanish"	"TranslateToSpanish"	"NorwegianText"	"SpanishText"
"GoogleTranslate"	"SpanishToEnglish"	"TranslateToEnglish"	"SpanishText"	"EnglishText"
"GoogleTranslate"	"SpanishToFrench"	"TranslateToFrench"	"SpanishText"	"FrenchText"
"GoogleTranslate"	"SpanishToHungarish"	"TranslateToHungarish"	"SpanishText"	"HungarishText"
"GoogleTranslate"	"SpanishToNorwegian"	"TranslateToNorwegian"	"SpanishText"	"NorwegianText"
"GaleGroupWebDomainBusinessIntelligence"	"GetFinancialInfo"	"GetFinancialInfoByUri"	"URI"	"FinancialInfo"
"GaleGroupBusinessIntelligence"	"GetFinancialInfo"	"GetFinancialInfoByCompanyName"	"CompanyName"	"FinancialInfo"
"ZipCodeDistance"	"GetLocation"	"GetLocationByZipCode"	"N/A"	"ZipCode"
"GeoIPLocation"	"GetLocation"	"GetLocationByIp"	"IPAddress"	"ZipCode"
"USWeatherForecast"	"WeatherForecast"	"WeatherForecastByUsCity"	"CityInUnitedStates"	"WeatherInfo"
"HistoricalStockDividends"	"GetStockInfo"	"GetStockInfo"	"StockTicker"	"StockInfo"
"CDYNEPhoneNotify"	"TextSender"	"SendText"	"PhoneNumber,Text"	"N/A"
"CDYNESMSNotify"	"SmsSender"	"SendSms"	"Text,SmsAddress"	"N/A"
"Reuters"	"GetNews"	"ProvideNews"	"N/A"	"News"
"InterestRateCalendar"	"GetInterestRate"	"GetInterestInfo"	"BondName"	"InterestInfo"
"PhoneBusinessInfo"	"PhoneToBusinessInfo"	"GetBusinessInfoByPhoneNumber"	"PhoneNumber"	"BusinessInfo"
"PhoneResidenceInfo"	"PhoneToResidenceInfo"	"GetResidenceInfoByPhoneNumber"	"PhoneNumber"	"ResidenceInfo"
"JobsInBritishColumbiaAndCanada"	"GetJobInfo"	"GetJobInfo"	"CityInBritish,CityInColumbia,CityInCanada"	"JobInfo"
"MightyMeals"	"Recipe"	"GetRecipe"	"MenuName"	"Recipe"
"YellowPageLookUp"	"GetCompanyInfoByName"	"GetCompanyInfoByName"	"CompanyName"	"CompanyInfo"
"Map"	"AddMaptoWebsite"	"AddMaptoWebsite"	"URL"	"Map"
"GoogleSearchWebService"	"CheckWebRanking"	"GetWebRanking"	"URL"	"WebRanking"
"BusinessTree"	"GetBusinessTree"	"GetBusinessTree"	"CompanyName"	"BusinessTree"
"VideoWebService"	"VideoSubscribe"	"VideoSubscribe"	"URL"	"Video"
"RSSWebService"	"RSSFeed"	"RssFeed"	"URL"	"RSSFeed"
"OutletAnalysisByCity"	"AnalyseOutlet"	"AnalyseOutlet"	"CityName"	"AnalyseOutletResult"
"OutletAnalysisByCountry"	"AnalyseOutlet"	"AnalyseOutlet"	"CountryName"	"AnalyseOutletResult"
"GeographicByZipcode"	"GetLocation"	"GetLocationByZipCode"	"ZipCode"	"GeographicalLocation"
"GlobalWeatherForecast"	"WeatherReport"	"WeatherReport"	"CityName"	"WeatherReport"
"WeatherForecastPro"	"WeatherReport"	"WeatherReport"	"CountryName"	"WeatherReport"
"FedExWebService"	"GetShippingRate"	"GetFedexShippingRate"	"CountryName"	"ShippingRate"
"TemperatureConversion"	"ConverseTemperature"	"ConverseTemperature"	"Celsius"	"Fahrenheit"
"CurrencyConverter"	"ConverseCurrency"	"ConverseCurrency"	"Currency"	"Currency"
"BICAndIBANValidation"	"ValidateBICOrIBAN"	"ValidateBicOrIban"	"BIC,IBAN"	"ValidationResult"
"GeoServiceTownland"	"GetGeographicInfo"	"GetGeographicInfoByCity"	"CityName"	"GeographicInfo"
"GeoServiceCountry"	"GetGeographicInfo"	"GetGeographicInfoByCountry"	"CountryName"	"GeographicInfo"
"GermanBankCodeLookup"	"SearchForBankCode"	"SearchForBankCode"	"BankName,CityName,Zipcode"	"BankCode"
"USZipValidator"	"GetPosition"	"GetPositionByUsZipCode"	"USZipCode"	"StateName,Latitude,Longitude"
"PhoneNumberValidation"	"ValidatePhoneNumber"	"ValidatePhoneNumber"	"USPhoneNumber,CanadaPhoneNumber"	"ValidationResult"
"GoogleSearch"	"SearchTheWeb"	"SearchTheWeb"	"Text"	"URL"
"FOREX"	"GetCurrencyExchangeRate"	"GetCurrencyExchangeRate"	"Currency"	"Currency"
"JadukaAPI"	"ProvideVoiceAPI"	"ProvideVoiceApi"	"FunctionCall"	"FunctionReturn"
"URLToIPAddress"	"GetIPAddress"	"GetIpAddress"	"URL"	"IPAddress"
"PortfolioSimulation"	"CalculateRangeOfReturn"	"CalculateRangeOfReturn"	"Portfolio"	"RangeOfReturn"
"XigniteCalendar"	"GetFinancialInfo"	"GetFinancialInfoInUSTreasury"	"USTreasury"	"FinancialInfo"
"LMEQuotes"	"GetPriceInfo"	"GetPriceInfoInLme"	"LMEStock"	"PriceInfo"
"NYBOTQuotes"	"GetPriceInfo"	"GetPriceInfoInNybot"	"NYBOTStock"	"PriceInfo"
"MGEXQuotes"	"GetPriceInfo"	"GetPriceInfoInMgex"	"MGEXStock"	"PriceInfo"
"WCEQuotes"	"GetPriceInfo"	"GetPriceInfoInWce"	"WCEStock"	"PriceInfo"
"KCBTQuotes"	"GetPriceInfo"	"GetPriceInfoInKcbt"	"KCBTStock"	"PriceInfo"
"COMEXQuotes"	"GetPriceInfo"	"GetPriceInfoInComex"	"COMEXStock"	"PriceInfo"
"CMEQuotes"	"GetPriceInfo"	"GetPriceInfoInCmeq"	"CMEQStock"	"PriceInfo"
"CBOTQuotes"	"GetPriceInfo"	"GetPriceInfoInCbot"	"CBOTStock"	"PriceInfo"
"NYMEXQuotes"	"GetPriceInfo"	"GetPriceInfoInNymex"	"NYMEXStock"	"PriceInfo"
"IBANValidator"	"ValidateIBAN"	"ValidateIban"	"IBAN"	"ValidateResult"

IA configuration files

"AmortizationCalculator"	"CalculateAmortization"	"CalculateAmortization"	"Princi-
pal,InterestRate,NumberOfPayment"	"Amortization"		
"CityAndStateByZIP"	"GetCityNameAndStateName"	"GetCityNameAndStateNameByZipCode"	"Zip-
Code" "CityNameAndStateName"			
"IPAddressLookup"	"GetIPAddress"	"GetIpAddress"	"URL" "IPAddress"
"BankVallInternational"	"ValidateSWIFTAndIBAN"	"ValidateSwiftAndIban"	"SWIFT,IBAN" "ValidationRe-
sult"			
"WeatherByCity"	"WeatherForecast"	"WeatherForecast"	"CityName" "WeatherInfo"
"ForecastByZipCode"	"WeatherForecast"	"WeatherForecastByZipCode"	"ZIPCode" "WeatherInfo"
"TelephoneVerification"	"VerifyTelephoneNumber"	"VerifyTelephoneNumber"	"PhoneNumber" "VerificationRe-
sult"			
"GBSCTHolidayDates"	"GetHolidayDates"	"GetHolidayDatesInScotland"	"CityInScotland" "HolidayDates"
"GBNIRHolidayDates"	"GetHolidayDates"	"GetHolidayDatesInNorthIreland"	"CityInNorthIreland" "Holi-
dayDates"			
"GBEAWHolidayDates"	"GetHolidayDates"	"GetHolidayDatesInGb"	"CityInEngland,CityInWales" "Holi-
dayDates"			
"HolidayDates"	"GetHolidayDates"	"GetHolidayDatesByCity"	"CityName" "HolidayDates"
"DigitalSignature"	"GetDigitalSignature"	"GetDigitalSignature"	"Key,Message" "DigitalSignature"
"DigitalSignatureVerify"	"VerifyDigitalSignature"	"VerifyDigitalSignature"	"Key,DigitalSignature,Message"
"VerificationResult"			
"ZipCodes"	"ObtainAllLocationInfo"	"ObtainAllLocationInfo"	"ZIPCode" "Longti-
tude,Latitude,CityName,StateName"			
"RestaurantFinding"	"SearchRestaurant"	"SearchRestaurantByName"	"RestaurantName" "RestaurantInfoEnglish"
"VEGRestaurantFinding"	"SearchRestaurant"	"SearchVegetarianRestaurant"	"Address" "RestaurantIn-
foEnglish"			
"CHIRestaurantFinding"	"SearchRestaurant"	"SearchChineseRestaurant"	"RestaurantName" "RestaurantInfoEnglish"
"ZIPRestaurantFinding"	"SearchRestaurant"	"SearchRestaurantByZipCode"	"ZipCode" "Restaurant-
Name,RestaurantInfoEnglish"			
"TYPRestaurantFinding"	"SearchRestaurant"	"SearchRestaurantByType"	"RestaurantType" "Restaurant-
Name,RestaurantInfoEnglish"			
"NORRestaurantFinding"	"SearchRestaurant"	"SearchNorwegianRestaurant"	"RestaurantName" "RestaurantIn-
foEnglish"			
"TheaterSearch"	"SearchTheater"	"SearchTheaterByAddress"	"Address" "TheaterName"
"CITYTheaterSearch"	"SearchTheater"	"SearchTheaterByCityName"	"CityName" "Theater-
Name,TheaterAddress"			
"STATTheaterSearch"	"SearchTheater"	"SearchTheaterByStateName"	"StateName" "Theater-
Name,TheaterAddress"			
"LANGTheaterSearch"	"SearchTheater"	"SearchTheaterByLanguage"	"Language" "Theater-
Name,TheaterAddress"			
"TIMTheaterSearch"	"SearchTheater"	"SearchTheaterByStartTime"	"StartTime" "Theater-
Name,TheaterAddress"			
"PRICERestaurantFinding"	"SearchRestaurant"	"SearchRestaurantByPrice"	"Price" "Restaurant-
Name,RestaurantAddress"			
"PRICTheaterSearch"	"SearchTheater"	"SearchTheaterByPrice"	"Price" "TheaterName,TheaterAddress"
"MapSearch"	"GetMap"	"GetMap"	"Address" "Map"
"MapFinding"	"GetMap"	"GetMap"	"Address" "Map"
"DistanceService"	"GetDistance"	"GetDistance"	"Address,Address" "Distance"
"ICQSender"	"SendText"	"SendIcqText"	"Text,IcqAddress" "N/A"
"SmsSender"	"SendText"	"SendSmsText"	"Text,PhoneNumber" "N/A"
"InstantMessageAlert"	"SendText"	"SendText"	"Text,IcqAddress" "N/A"
"SASBooking"	"BookFlight"	"BookSasFlight"	"SasFlightName,DepartureDate" "N/A"
"SterlingBooking"	"BookFlight"	"BookSterlingFlight"	"SterlingFlightName,DepartureDate" "N/A"
"SkyScanner"	"SearchFlight"	"SearchFlight"	"DepartCity,ArriveCity,DepartureDate" "Flight-
Name,DepartureDate,DepartureTime"			
"FlightBooking"	"BookFlight"	"BookFlight"	"FlightName,DepartureDate,DepartureTime" "N/A"
"SpellCheck"	"CheckSpelling"	"CheckSpelling"	"Text" "Text"
"UKLocation"	"GetLocation"	"GetLocation"	"ZipCode" "GeographicalLocation"
"UKLocationPro"	"GetLocation"	"GetLocation"	"CellID" "GeographicalLocation"
"QuoteOfTheDay"	"GetQuote"	"GetQuote"	"N/A" "Quote"
"RestaurantBooking"	"Book"	"Book"	"RestaurantName" "N/A"
"CITYTrafficReport"	"ReportTrafficCondition"	"GetTrafficCondition"	"CityName" "TrafficReport"
"StreetTrafficReport"	"ReportTrafficCondition"	"GetTrafficCondition"	"StreetName" "TrafficReport"
"USTrafficReport"	"ReportTrafficCondition"	"GetTrafficCondition"	"USStreetName" "TrafficReport"
"GlobalSMS"	"SendSms"	"SendSms"	"Text,PhoneNumber" "N/A"
"GlobalNews"	"SendNews"	"SendNews"	"PhoneNumber" "N/A"
"MailLocate"	"GetEmail"	"GetEmail"	"Name" "EmailAddress"

IA configuration files

"MailDaemon"	"SendEmail"	"SendEmail"	"Text,EmailAddress"	"N/A"
"EmailDelivery"	"SendEmail"	"SendEmail"	"Text,EmailAddress"	"N/A"
"ISOCCode"	"GetISOCCode"	"GetISOCCode"	"CityName"	"ISOCCode"
"FaxService"	"SendFax"	"SendFax"	"PhoneNumber,Text"	"N/A"
"GetJoke"	"GetJoke"	"GetJoke"	"N/A"	"Joke"
"AirportInfo"	"GetAirportInfo"	"GetAirportInfo"	"AirportName"	"AirportInfo"
"Lotto"	"GenerateRandomNumber"	"GenerateRandomNumber"	"N/A"	"Number"
"GeoPlaces"	"GetGeoDistance"	"GetGeoDistance"	"ZipCode,ZipCode"	"Distance"
"LocalTime"	"ReturnLocalTime"	"ReturnLocalTime"	"ZipCode"	"Time"
"FindMp3"	"FindMp3"	"FindMp3"	"Mp3Name"	"Mp3"
"END"				

C.2 Goal Ontology (Goals.owl)

```
<?xml version="1.0"?>
<rdf:RDF
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:lw="http://ontosphere3d.sourceforge.net/LogicViews.owl#"
  xmlns="http://www.spicerepository.net/Goals.owl#"
  xmlns:owl="http://www.w3.org/2002/07/owl#"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema#"
  xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
  xml:base="http://www.spicerepository.net/Goals.owl">
  <owl:Ontology rdf:about=""/>
  <owl:Class rdf:ID="GetPriceInfoInNymex">
    <rdfs:subClassOf>
      <owl:Class rdf:ID="GetPriceInfo"/>
    </rdfs:subClassOf>
  </owl:Class>
  <owl:Class rdf:ID="GetGeoDistance">
    <rdfs:subClassOf>
      <owl:Class rdf:ID="GetDistance"/>
    </rdfs:subClassOf>
  </owl:Class>
  <owl:Class rdf:ID="GetResidenceInfoByPhoneNumber">
    <rdfs:subClassOf>
      <owl:Class rdf:ID="GetResidenceInfo"/>
    </rdfs:subClassOf>
  </owl:Class>
  <owl:Class rdf:ID="GetGeographicInfoByCountry">
    <rdfs:subClassOf>
      <owl:Class rdf:ID="GetGeographicInfo"/>
    </rdfs:subClassOf>
  </owl:Class>
  <owl:Class rdf:ID="GetEmail">
    <rdfs:subClassOf>
      <owl:Class rdf:ID="Get"/>
    </rdfs:subClassOf>
  </owl:Class>
  <owl:Class rdf:ID="SendInformation">
    <rdfs:subClassOf>
      <owl:Class rdf:ID="Send"/>
    </rdfs:subClassOf>
  </owl:Class>
  <owl:Class rdf:about="#GetResidenceInfo">
    <rdfs:subClassOf>
      <owl:Class rdf:ID="GetInfo"/>
    </rdfs:subClassOf>
  </owl:Class>
  <owl:Class rdf:about="#GetInfo">
    <rdfs:subClassOf>
      <owl:Class rdf:about="#Get"/>
    </rdfs:subClassOf>
  </owl:Class>
  <owl:Class rdf:ID="Goals"/>
  <owl:Class rdf:ID="Find"/>
</rdf:RDF>
```

IA configuration files

```
<rdfs:subClassOf rdf:resource="#Goals"/>
</owl:Class>
<owl:Class rdf:ID="SearchRestaurantByZipCode">
  <rdfs:subClassOf>
    <owl:Class rdf:ID="FindRestaurant"/>
  </rdfs:subClassOf>
</owl:Class>
<owl:Class rdf:ID="FindRestaurantByLocation">
  <rdfs:subClassOf>
    <owl:Class rdf:about="#FindRestaurant"/>
  </rdfs:subClassOf>
</owl:Class>
<owl:Class rdf:about="#Send">
  <rdfs:subClassOf rdf:resource="#Goals"/>
</owl:Class>
<owl:Class rdf:ID="GetDigitalSignature">
  <rdfs:subClassOf>
    <owl:Class rdf:about="#Get"/>
  </rdfs:subClassOf>
</owl:Class>
<owl:Class rdf:ID="GetLocationByUSZipCode">
  <rdfs:subClassOf>
    <owl:Class rdf:ID="GetLocation"/>
  </rdfs:subClassOf>
</owl:Class>
<owl:Class rdf:ID="GetGeographicInfoByCity">
  <rdfs:subClassOf>
    <owl:Class rdf:about="#GetGeographicInfo"/>
  </rdfs:subClassOf>
</owl:Class>
<owl:Class rdf:ID="GetRecipe">
  <rdfs:subClassOf>
    <owl:Class rdf:about="#Get"/>
  </rdfs:subClassOf>
</owl:Class>
<owl:Class rdf:ID="FindMp3">
  <rdfs:subClassOf rdf:resource="#Find"/>
</owl:Class>
<owl:Class rdf:ID="GetInterestInfo">
  <rdfs:subClassOf rdf:resource="#GetInfo"/>
</owl:Class>
<owl:Class rdf:about="#GetGeographicInfo">
  <rdfs:subClassOf rdf:resource="#GetInfo"/>
</owl:Class>
<owl:Class rdf:ID="SearchTheater">
  <rdfs:subClassOf rdf:resource="#Find"/>
</owl:Class>
<owl:Class rdf:ID="SearchFlight">
  <rdfs:subClassOf rdf:resource="#Find"/>
</owl:Class>
<owl:Class rdf:ID="SearchForBankCode">
  <rdfs:subClassOf rdf:resource="#Find"/>
</owl:Class>
<owl:Class rdf:ID="BookSterlingFlight">
  <rdfs:subClassOf>
    <owl:Class rdf:ID="BookFlight"/>
  </rdfs:subClassOf>
</owl:Class>
<owl:Class rdf:ID="GetStockInfo">
  <rdfs:subClassOf rdf:resource="#GetInfo"/>
</owl:Class>
<owl:Class rdf:ID="SearchVegetarianRestaurant">
  <rdfs:subClassOf>
    <owl:Class rdf:about="#FindRestaurant"/>
  </rdfs:subClassOf>
</owl:Class>
<owl:Class rdf:ID="SearchChineseRestaurant">
```

IA configuration files

```
<rdfs:subClassOf>
  <owl:Class rdf:about="#FindRestaurant"/>
</rdfs:subClassOf>
</owl:Class>
<owl:Class rdf:ID="GetHolidayDatesByCountry">
  <rdfs:subClassOf>
    <owl:Class rdf:ID="GetHolidayDates"/>
  </rdfs:subClassOf>
</owl:Class>
<owl:Class rdf:ID="GetPriceInfoInCbot">
  <rdfs:subClassOf>
    <owl:Class rdf:about="#GetPriceInfo"/>
  </rdfs:subClassOf>
</owl:Class>
<owl:Class rdf:ID="TranslateToHungarish">
  <rdfs:subClassOf>
    <owl:Class rdf:ID="Translate"/>
  </rdfs:subClassOf>
</owl:Class>
<owl:Class rdf:ID="GetMap">
  <rdfs:subClassOf>
    <owl:Class rdf:about="#Get"/>
  </rdfs:subClassOf>
</owl:Class>
<owl:Class rdf:ID="GetFinacialInfoByUri">
  <rdfs:subClassOf>
    <owl:Class rdf:ID="GetFinancialInfo"/>
  </rdfs:subClassOf>
</owl:Class>
<owl:Class rdf:about="#Get">
  <rdfs:subClassOf rdf:resource="#Goals"/>
</owl:Class>
<owl:Class rdf:ID="ProvideVoiceApi">
  <rdfs:subClassOf>
    <owl:Class rdf:ID="GetApi"/>
  </rdfs:subClassOf>
</owl:Class>
<owl:Class rdf:about="#GetHolidayDates">
  <rdfs:subClassOf rdf:resource="#Get"/>
</owl:Class>
<owl:Class rdf:ID="GetWebRanking">
  <rdfs:subClassOf rdf:resource="#Get"/>
</owl:Class>
<owl:Class rdf:ID="GetTrafficCondition">
  <rdfs:subClassOf rdf:resource="#Get"/>
</owl:Class>
<owl:Class rdf:ID="GetCurrencyExchangeRate">
  <rdfs:subClassOf rdf:resource="#Get"/>
</owl:Class>
<owl:Class rdf:about="#FindRestaurant">
  <rdfs:subClassOf rdf:resource="#Find"/>
</owl:Class>
<owl:Class rdf:about="#GetFinancialInfo">
  <rdfs:subClassOf rdf:resource="#GetInfo"/>
</owl:Class>
<owl:Class rdf:ID="SendSmsText">
  <rdfs:subClassOf rdf:resource="#Send"/>
</owl:Class>
<owl:Class rdf:ID="SearchNorwegianRestaurant">
  <rdfs:subClassOf rdf:resource="#FindRestaurant"/>
</owl:Class>
<owl:Class rdf:ID="Book">
  <rdfs:subClassOf rdf:resource="#Goals"/>
</owl:Class>
<owl:Class rdf:ID="GetHolidayDatesInScotland">
  <rdfs:subClassOf rdf:resource="#GetHolidayDatesByCountry"/>
</owl:Class>
```


IA configuration files

```
<owl:Class rdf:ID="GetPriceInfoInKcibt">
  <rdfs:subClassOf>
    <owl:Class rdf:about="#GetPriceInfo"/>
  </rdfs:subClassOf>
</owl:Class>
<owl:Class rdf:ID="GetIsoCode">
  <rdfs:subClassOf rdf:resource="#Get"/>
</owl:Class>
<owl:Class rdf:ID="GetCityNameAndStateNameByZipCode">
  <rdfs:subClassOf>
    <owl:Class rdf:ID="GetCityNameAndStateName"/>
  </rdfs:subClassOf>
</owl:Class>
<owl:Class rdf:ID="GetPriceInfoInLme">
  <rdfs:subClassOf>
    <owl:Class rdf:about="#GetPriceInfo"/>
  </rdfs:subClassOf>
</owl:Class>
<owl:Class rdf:ID="TranslateToEnglish">
  <rdfs:subClassOf>
    <owl:Class rdf:about="#Translate"/>
  </rdfs:subClassOf>
</owl:Class>
<owl:Class rdf:ID="SearchRestaurantByType">
  <rdfs:subClassOf rdf:resource="#FindRestaurant"/>
</owl:Class>
<owl:Class rdf:ID="WeatherReport">
  <rdfs:subClassOf rdf:resource="#Forecast"/>
</owl:Class>
<owl:Class rdf:ID="GetFedexShippingRate">
  <rdfs:subClassOf rdf:resource="#Get"/>
</owl:Class>
<owl:Class rdf:about="#Report">
  <rdfs:subClassOf rdf:resource="#Goals"/>
</owl:Class>
<owl:Class rdf:ID="WeatherForecastByZipCode">
  <rdfs:subClassOf>
    <owl:Class rdf:ID="WeatherForecast"/>
  </rdfs:subClassOf>
</owl:Class>
<owl:Class rdf:ID="GetFinancialInfoByCompanyName">
  <rdfs:subClassOf rdf:resource="#GetFinancialInfo"/>
</owl:Class>
<owl:Class rdf:about="#GetApi">
  <rdfs:subClassOf rdf:resource="#Get"/>
</owl:Class>
<owl:Class rdf:ID="SearchTheaterByLangauge">
  <rdfs:subClassOf rdf:resource="#SearchTheater"/>
</owl:Class>
<owl:Class rdf:about="#GetPriceInfo">
  <rdfs:subClassOf rdf:resource="#GetFinancialInfo"/>
</owl:Class>
<owl:Class rdf:ID="ProvideNews">
  <rdfs:subClassOf>
    <owl:Class rdf:ID="GetNews"/>
  </rdfs:subClassOf>
</owl:Class>
<owl:Class rdf:ID="GetUserInformation">
  <rdfs:subClassOf rdf:resource="#GetInfo"/>
</owl:Class>
<owl:Class rdf:about="#GetNews">
  <rdfs:subClassOf rdf:resource="#Get"/>
</owl:Class>
<owl:Class rdf:ID="Forecast">
  <rdfs:subClassOf rdf:resource="#Goals"/>
</owl:Class>
<owl:Class rdf:ID="SendSms">
```

IA configuration files

```
<rdfs:subClassOf rdf:resource="#Send"/>
</owl:Class>
<owl:Class rdf:ID="SendEmail">
  <rdfs:subClassOf rdf:resource="#Send"/>
</owl:Class>
<owl:Class rdf:ID="GetUkLocation">
  <rdfs:subClassOf>
    <owl:Class rdf:about="#GetLocation"/>
  </rdfs:subClassOf>
</owl:Class>
<owl:Class rdf:ID="FindRestaurantInFrance">
  <rdfs:subClassOf rdf:resource="#FindRestaurant"/>
</owl:Class>
<owl:Class rdf:about="#Translate">
  <rdfs:subClassOf rdf:resource="#Goals"/>
</owl:Class>
<owl:Class rdf:about="#WeatherForecast">
  <rdfs:subClassOf rdf:resource="#Forecast"/>
</owl:Class>
<owl:Class rdf:ID="TranslateToFrench">
  <rdfs:subClassOf rdf:resource="#Translate"/>
</owl:Class>
<owl:Class rdf:ID="GetLocationByIp">
  <rdfs:subClassOf>
    <owl:Class rdf:about="#GetLocation"/>
  </rdfs:subClassOf>
</owl:Class>
<owl:Class rdf:ID="GetJoke">
  <rdfs:subClassOf rdf:resource="#Get"/>
</owl:Class>
<owl:Class rdf:ID="SearchTheaterByAddress">
  <rdfs:subClassOf rdf:resource="#SearchTheater"/>
</owl:Class>
<owl:Class rdf:ID="SearchTheaterByPrice">
  <rdfs:subClassOf rdf:resource="#SearchTheater"/>
</owl:Class>
<owl:Class rdf:ID="GetBusinessInfo">
  <rdfs:subClassOf rdf:resource="#GetInfo"/>
</owl:Class>
<owl:Class rdf:ID="GetHolidayDatesInGb">
  <rdfs:subClassOf rdf:resource="#GetHolidayDatesByCountry"/>
</owl:Class>
<owl:Class rdf:ID="GetPriceInfoInWce">
  <rdfs:subClassOf rdf:resource="#GetPriceInfo"/>
</owl:Class>
<owl:Class rdf:ID="GetQuote">
  <rdfs:subClassOf rdf:resource="#Get"/>
</owl:Class>
<owl:Class rdf:ID="BookRestaurant">
  <rdfs:subClassOf rdf:resource="#Book"/>
</owl:Class>
<owl:Class rdf:ID="GetHolidayDatesByCity">
  <rdfs:subClassOf rdf:resource="#GetHolidayDates"/>
</owl:Class>
<owl:Class rdf:ID="SearchTheaterByCityName">
  <rdfs:subClassOf rdf:resource="#SearchTheater"/>
</owl:Class>
<owl:Class rdf:about="#GetDistance">
  <rdfs:subClassOf rdf:resource="#Get"/>
</owl:Class>
<owl:Class rdf:ID="SendFax">
  <rdfs:subClassOf rdf:resource="#Send"/>
</owl:Class>
<owl:Class rdf:ID="TranslateToNorwegian">
  <rdfs:subClassOf rdf:resource="#Translate"/>
</owl:Class>
<owl:Class rdf:ID="SearchRestaurantByPrice">
```

IA configuration files

```
<rdfs:subClassOf rdf:resource="#FindRestaurant"/>
</owl:Class>
<owl:Class rdf:ID="FindAddressOfLocation">
  <rdfs:subClassOf>
    <owl:Class rdf:ID="FindAddress"/>
  </rdfs:subClassOf>
</owl:Class>
<owl:Class rdf:ID="FindRestaurantByAddress">
  <rdfs:subClassOf rdf:resource="#FindRestaurant"/>
</owl:Class>
<owl:Class rdf:ID="GetAreaCode">
  <rdfs:subClassOf rdf:resource="#Get"/>
</owl:Class>
<owl:Class rdf:ID="GetPriceInfoInComex">
  <rdfs:subClassOf rdf:resource="#GetPriceInfo"/>
</owl:Class>
<owl:Class rdf:ID="GetJobInfo">
  <rdfs:subClassOf rdf:resource="#GetInfo"/>
</owl:Class>
<owl:Class rdf:about="#GetLocation">
  <rdfs:subClassOf rdf:resource="#Get"/>
</owl:Class>
<owl:Class rdf:ID="GetAirportInfo">
  <rdfs:subClassOf rdf:resource="#GetInfo"/>
</owl:Class>
<owl:Class rdf:ID="FindRestaurantByLocationAndType">
  <rdfs:subClassOf rdf:resource="#FindRestaurantByLocation"/>
  <rdfs:subClassOf rdf:resource="#SearchRestaurantByType"/>
</owl:Class>
<owl:Class rdf:ID="SendIcqText">
  <rdfs:subClassOf rdf:resource="#Send"/>
</owl:Class>
<owl:Class rdf:ID="BookSasFlight">
  <rdfs:subClassOf>
    <owl:Class rdf:about="#BookFlight"/>
  </rdfs:subClassOf>
</owl:Class>
<owl:Class rdf:about="#BookFlight">
  <rdfs:subClassOf rdf:resource="#Book"/>
</owl:Class>
<owl:Class rdf:ID="GetPriceInfoInMgex">
  <rdfs:subClassOf rdf:resource="#GetPriceInfo"/>
</owl:Class>
<owl:Class rdf:ID="GetFinancialInfoInUsTreasury">
  <rdfs:subClassOf rdf:resource="#GetFinancialInfo"/>
</owl:Class>
<owl:Class rdf:ID="GetHolidayDatesInNorthIreland">
  <rdfs:subClassOf rdf:resource="#GetHolidayDatesByCountry"/>
</owl:Class>
<owl:Class rdf:about="#GetCityNameAndStateName">
  <rdfs:subClassOf rdf:resource="#GetLocation"/>
</owl:Class>
<owl:Class rdf:ID="SendMms">
  <rdfs:subClassOf rdf:resource="#Send"/>
</owl:Class>
<owl:Class rdf:ID="GetPriceInfoInNybot">
  <rdfs:subClassOf rdf:resource="#GetPriceInfo"/>
</owl:Class>
<owl:Class rdf:ID="SearchTheaterByStartTime">
  <rdfs:subClassOf rdf:resource="#SearchTheater"/>
</owl:Class>
<owl:Class rdf:ID="GetCompanyInfoByName">
  <rdfs:subClassOf>
    <owl:Class rdf:ID="GetCompanyInfo"/>
  </rdfs:subClassOf>
</owl:Class>
<owl:Class rdf:about="#GetCompanyInfo">
```

IA configuration files

```
<rdf:subClassOf rdf:resource="#GetInfo"/>
</owl:Class>
<owl:Class rdf:ID="WeatherForecastByUsCity">
  <rdf:subClassOf rdf:resource="#WeatherForecast"/>
</owl:Class>
<owl:Class rdf:ID="SearchTheaterByStateName">
  <rdf:subClassOf rdf:resource="#SearchTheater"/>
</owl:Class>
<owl:Class rdf:ID="GetIpAddress">
  <rdf:subClassOf rdf:resource="#Get"/>
</owl:Class>
<owl:Class rdf:ID="GetPriceInfoInCmeq">
  <rdf:subClassOf rdf:resource="#GetPriceInfo"/>
</owl:Class>
<owl:Class rdf:ID="GetBusinessInfoByPhoneNumber">
  <rdf:subClassOf rdf:resource="#GetBusinessInfo"/>
</owl:Class>
<owl:Class rdf:ID="GetBusinessTree">
  <rdf:subClassOf rdf:resource="#GetBusinessInfo"/>
</owl:Class>
<owl:Class rdf:ID="TranslateToSpanish">
  <rdf:subClassOf rdf:resource="#Translate"/>
</owl:Class>
<owl:Class rdf:ID="SearchRestaurantByName">
  <rdf:subClassOf rdf:resource="#FindRestaurant"/>
</owl:Class>
<owl:Class rdf:ID="GetLocationByZipCode">
  <rdf:subClassOf rdf:resource="#GetLocation"/>
</owl:Class>
<owl:Class rdf:about="#FindAddress">
  <rdf:subClassOf rdf:resource="#Find"/>
</owl:Class>
<owl:ObjectProperty rdf:ID="isInputOf">
  <rdf:domain>
    <owl:Class>
      <owl:unionOf rdf:parseType="Collection">
        <owl:Class rdf:about="#FindRestaurant"/>
        <owl:Class rdf:about="#FindAddress"/>
        <owl:Class rdf:about="#GetCityNameAndStateName"/>
        <owl:Class rdf:about="#GetIpAddress"/>
        <owl:Class rdf:about="#GetGeographicInfo"/>
      </owl:unionOf>
    </owl:Class>
  </rdf:domain>
  <rdf:range>
    <owl:Class>
      <owl:unionOf rdf:parseType="Collection">
        <owl:Class rdf:about="#BookRestaurant"/>
        <owl:Class rdf:about="#SearchTheater"/>
        <owl:Class rdf:about="#WeatherForecast"/>
        <owl:Class rdf:about="#WeatherReport"/>
        <owl:Class rdf:about="#GetGeographicInfo"/>
        <owl:Class rdf:about="#GetLocationByIp"/>
        <owl:Class rdf:about="#GetHolidayDatesByCity"/>
        <owl:Class rdf:about="#GetMap"/>
        <owl:Class rdf:about="#GetDistance"/>
        <owl:Class rdf:about="#GetAreaCode"/>
      </owl:unionOf>
    </owl:Class>
  </rdf:range>
</owl:ObjectProperty>
<WeatherForecast rdf:ID="WeatherForecastI"/>
<FindAddress rdf:ID="FindAddressI">
  <isInputOf>
    <SearchTheater rdf:ID="SearchTheaterI"/>
  </isInputOf>
</isInputOf>
```

IA configuration files

```
<GetMap rdf:ID="GetMapI"/>
</isInputOf>
</FindAddress>
<owl:AnnotationProperty rdf:about="http://ontosphere3d.sourceforge.net/LogicViews.owl#belongsToView">
  <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#DatatypeProperty"/>
</owl:AnnotationProperty>
<GetLocationByIP rdf:ID="GetLocationByIPi"/>
<GetAreaCode rdf:ID="GetAreaCodeI"/>
<BookRestaurant rdf:ID="BookRestaurantI"/>
<FindRestaurant rdf:ID="FindRestaurantI">
  <isInputOf rdf:resource="#BookRestaurantI"/>
</FindRestaurant>
<GetIPAddress rdf:ID="GetIPAddressI">
  <isInputOf rdf:resource="#GetLocationByIPi"/>
</GetIPAddress>
<GetHolidayDatesByCity rdf:ID="GetHolidayDatesByCityI"/>
<WeatherReport rdf:ID="WeatherReportI"/>
<GetCityNameAndStateName rdf:ID="GetCityNameAndStateNameI">
  <isInputOf rdf:resource="#WeatherForecastI"/>
  <isInputOf rdf:resource="#WeatherReportI"/>
  <isInputOf>
    <GetGeographicInfo rdf:ID="GetGeographicInfoI">
      <isInputOf>
        <GetDistance rdf:ID="GetDistanceI"/>
      </isInputOf>
    </GetGeographicInfo>
  </isInputOf>
  <isInputOf rdf:resource="#GetHolidayDatesByCityI"/>
  <isInputOf rdf:resource="#GetAreaCodeI"/>
</GetCityNameAndStateName>
<owl:AnnotationProperty rdf:about="http://ontosphere3d.sourceforge.net/LogicViews.owl#hasColor">
  <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#DatatypeProperty"/>
</owl:AnnotationProperty>
</rdf:RDF>

<!-- Created with Protege (with OWL Plugin 3.3.1, Build 430) http://protege.stanford.edu -->
```

C.3 Input/Output Ontology (IO.owl)

```
<?xml version="1.0"?>
<rdf:RDF
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns="http://www.spicerepository.net/IOTypes.owl#"
  xmlns:owl="http://www.w3.org/2002/07/owl#"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema#"
  xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
  xml:base="http://www.spicerepository.net/IOTypes.owl">
  <owl:Ontology rdf:about=""/>
  <owl:Class rdf:ID="Time">
    <rdfs:subClassOf>
      <owl:Class rdf:ID="IO"/>
    </rdfs:subClassOf>
  </owl:Class>
  <owl:Class rdf:ID="AirportInfo">
    <rdfs:subClassOf>
      <owl:Class rdf:ID="Text"/>
    </rdfs:subClassOf>
  </owl:Class>
  <owl:Class rdf:ID="MGEXStock">
    <rdfs:subClassOf>
      <owl:Class rdf:ID="StockName"/>
    </rdfs:subClassOf>
  </owl:Class>
  <owl:Class rdf:ID="LMESTock">
    <rdfs:subClassOf>
      <owl:Class rdf:about="#StockName"/>
    </rdfs:subClassOf>
  </owl:Class>
```

```
</rdfs:subClassOf>
</owl:Class>
<owl:Class rdf:ID="CityInNorthIreland">
  <rdfs:subClassOf>
    <owl:Class rdf:ID="CityName"/>
  </rdfs:subClassOf>
</owl:Class>
<owl:Class rdf:ID="CityInUnitedStates">
  <rdfs:subClassOf>
    <owl:Class rdf:about="#CityName"/>
  </rdfs:subClassOf>
</owl:Class>
<owl:Class rdf:ID="NYBOTStock">
  <rdfs:subClassOf>
    <owl:Class rdf:about="#StockName"/>
  </rdfs:subClassOf>
</owl:Class>
<owl:Class rdf:ID="MenuName">
  <rdfs:subClassOf rdf:resource="#IO"/>
</owl:Class>
<owl:Class rdf:ID="CityInWales">
  <rdfs:subClassOf>
    <owl:Class rdf:about="#CityName"/>
  </rdfs:subClassOf>
</owl:Class>
<owl:Class rdf:ID="CityInFrance">
  <rdfs:subClassOf>
    <owl:Class rdf:about="#CityName"/>
  </rdfs:subClassOf>
</owl:Class>
<owl:Class rdf:ID="DepartCity">
  <rdfs:subClassOf>
    <owl:Class rdf:about="#CityName"/>
  </rdfs:subClassOf>
</owl:Class>
<owl:Class rdf:ID="ArriveCity">
  <rdfs:subClassOf>
    <owl:Class rdf:about="#CityName"/>
  </rdfs:subClassOf>
</owl:Class>
<owl:Class rdf:ID="GeographicalLocation">
  <rdfs:subClassOf>
    <owl:Class rdf:ID="Location"/>
  </rdfs:subClassOf>
  <rdfs:subClassOf>
    <owl:Class rdf:about="#Text"/>
  </rdfs:subClassOf>
</owl:Class>
<owl:Class rdf:ID="WeatherInfo">
  <rdfs:subClassOf>
    <owl:Class rdf:about="#Text"/>
  </rdfs:subClassOf>
</owl:Class>
<owl:Class rdf:ID="Distance">
  <rdfs:subClassOf>
    <owl:Class rdf:about="#Text"/>
  </rdfs:subClassOf>
</owl:Class>
<owl:Class rdf:about="#Location">
  <rdfs:subClassOf rdf:resource="#IO"/>
</owl:Class>
<owl:Class rdf:ID="FlightName">
  <rdfs:subClassOf rdf:resource="#IO"/>
</owl:Class>
<owl:Class rdf:ID="CMEQStock">
  <rdfs:subClassOf>
    <owl:Class rdf:about="#StockName"/>
  </rdfs:subClassOf>
</owl:Class>
```

IA configuration files

```
</rdfs:subClassOf>
</owl:Class>
<owl:Class rdf:ID="WeatherReport">
  <rdfs:subClassOf>
    <owl:Class rdf:about="#Text"/>
  </rdfs:subClassOf>
</owl:Class>
<owl:Class rdf:ID="TrafficReport">
  <rdfs:subClassOf>
    <owl:Class rdf:about="#Text"/>
  </rdfs:subClassOf>
</owl:Class>
<owl:Class rdf:ID="Joke">
  <rdfs:subClassOf rdf:resource="#IO"/>
</owl:Class>
<owl:Class rdf:ID="RestaurantInfoEnglish">
  <rdfs:subClassOf>
    <owl:Class rdf:ID="EnglishText"/>
  </rdfs:subClassOf>
</owl:Class>
<owl:Class rdf:ID="CityInScotland">
  <rdfs:subClassOf>
    <owl:Class rdf:about="#CityName"/>
  </rdfs:subClassOf>
</owl:Class>
<SmsAddress rdf:ID="me_sms">
  <rdf:type rdf:resource="#PhoneNumber"/>
  <rdf:type rdf:resource="#MmsAddress"/>
</SmsAddress>
<owl:Class rdf:ID="UKaddress">
  <rdfs:subClassOf>
    <owl:Class rdf:about="#Location"/>
  </rdfs:subClassOf>
</owl:Class>
<owl:Class rdf:ID="Latitude">
  <rdfs:subClassOf rdf:resource="#Location"/>
</owl:Class>
<owl:Class rdf:ID="StockTicker">
  <rdfs:subClassOf rdf:resource="#IO"/>
</owl:Class>
<owl:Class rdf:ID="State">
  <rdfs:subClassOf rdf:resource="#Location"/>
</owl:Class>
<owl:Class rdf:ID="RestaurantInfoFrench">
  <rdfs:subClassOf>
    <owl:Class rdf:ID="FrenchText"/>
  </rdfs:subClassOf>
</owl:Class>
<owl:Class rdf:ID="Name">
  <rdfs:subClassOf rdf:resource="#IO"/>
</owl:Class>
<owl:Class rdf:ID="SasFlightName">
  <rdfs:subClassOf rdf:resource="#FlightName"/>
</owl:Class>
<owl:Class rdf:ID="CityInColumbia">
  <rdfs:subClassOf>
    <owl:Class rdf:about="#CityName"/>
  </rdfs:subClassOf>
</owl:Class>
<owl:Class rdf:ID="CBOTStock">
  <rdfs:subClassOf>
    <owl:Class rdf:about="#StockName"/>
  </rdfs:subClassOf>
</owl:Class>
<owl:Class rdf:ID="NorwegianText">
  <rdfs:subClassOf>
    <owl:Class rdf:about="#Text"/>
  </rdfs:subClassOf>
</owl:Class>
```

```
</rdfs:subClassOf>
</owl:Class>
<owl:Class rdf:about="#Text">
  <rdfs:subClassOf rdf:resource="#IO"/>
</owl:Class>
<owl:Class rdf:ID="Media_Content">
  <rdfs:subClassOf rdf:resource="#IO"/>
</owl:Class>
<owl:Class rdf:ID="TheaterAddress">
  <rdfs:subClassOf>
    <owl:Class rdf:ID="Address"/>
  </rdfs:subClassOf>
</owl:Class>
<owl:Class rdf:ID="CityInCanada">
  <rdfs:subClassOf>
    <owl:Class rdf:about="#CityName"/>
  </rdfs:subClassOf>
</owl:Class>
<owl:Class rdf:ID="SmsAddress">
  <rdfs:subClassOf>
    <owl:Class rdf:about="#ID"/>
  </rdfs:subClassOf>
</owl:Class>
<owl:Class rdf:about="#FrenchText">
  <rdfs:subClassOf rdf:resource="#Text"/>
</owl:Class>
<owl:Class rdf:ID="SterlingFlightName">
  <rdfs:subClassOf rdf:resource="#FlightName"/>
</owl:Class>
<owl:Class rdf:ID="MovieName">
  <rdfs:subClassOf rdf:resource="#IO"/>
</owl:Class>
<owl:Class rdf:ID="CanadaPhoneNumber">
  <rdfs:subClassOf>
    <owl:Class rdf:ID="PhoneNumber"/>
  </rdfs:subClassOf>
</owl:Class>
<owl:Class rdf:ID="RestaurantName">
  <rdfs:subClassOf rdf:resource="#IO"/>
  <rdfs:subClassOf rdf:resource="#Text"/>
</owl:Class>
<owl:Class rdf:ID="URL">
  <rdfs:subClassOf>
    <owl:Class rdf:about="#ID"/>
  </rdfs:subClassOf>
</owl:Class>
<owl:Class rdf:ID="YahooAddress">
  <rdfs:subClassOf>
    <owl:Class rdf:about="#ID"/>
  </rdfs:subClassOf>
</owl:Class>
<owl:Class rdf:ID="Number">
  <rdfs:subClassOf rdf:resource="#IO"/>
</owl:Class>
<owl:Class rdf:about="#Address">
  <rdfs:subClassOf rdf:resource="#Location"/>
  <rdfs:subClassOf rdf:resource="#Text"/>
</owl:Class>
<owl:Class rdf:ID="StockInfo">
  <rdfs:subClassOf rdf:resource="#Text"/>
</owl:Class>
<owl:Class rdf:ID="IMSI">
  <rdfs:subClassOf rdf:resource="#Location"/>
</owl:Class>
<owl:Class rdf:about="#EnglishText">
  <rdfs:subClassOf rdf:resource="#Text"/>
</owl:Class>
```


IA configuration files

```
<owl:Class rdf:ID="CountryName">
  <rdfs:subClassOf rdf:resource="#IO"/>
</owl:Class>
<owl:Class rdf:ID="AreaCode">
  <rdfs:subClassOf rdf:resource="#Location"/>
  <rdfs:subClassOf rdf:resource="#Text"/>
</owl:Class>
<owl:Class rdf:ID="ISOCODE">
  <rdfs:subClassOf rdf:resource="#Location"/>
</owl:Class>
<owl:Class rdf:ID="BondName">
  <rdfs:subClassOf rdf:resource="#IO"/>
</owl:Class>
<owl:Class rdf:ID="StreetName">
  <rdfs:subClassOf rdf:resource="#IO"/>
</owl:Class>
<owl:Class rdf:ID="USZipCode">
  <rdfs:subClassOf>
    <owl:Class rdf:ID="ZipCode"/>
  </rdfs:subClassOf>
</owl:Class>
<owl:Class rdf:ID="KCBTStock">
  <rdfs:subClassOf>
    <owl:Class rdf:about="#StockName"/>
  </rdfs:subClassOf>
</owl:Class>
<owl:Class rdf:ID="Map">
  <rdfs:subClassOf rdf:resource="#Media_Content"/>
</owl:Class>
<owl:Class rdf:ID="HungarishText">
  <rdfs:subClassOf rdf:resource="#Text"/>
</owl:Class>
<owl:Class rdf:ID="NYMEXStock">
  <rdfs:subClassOf>
    <owl:Class rdf:about="#StockName"/>
  </rdfs:subClassOf>
</owl:Class>
<owl:Class rdf:ID="CityInBritish">
  <rdfs:subClassOf>
    <owl:Class rdf:about="#CityName"/>
  </rdfs:subClassOf>
</owl:Class>
<owl:Class rdf:about="#ZipCode">
  <rdfs:subClassOf rdf:resource="#Location"/>
  <rdfs:subClassOf rdf:resource="#Address"/>
</owl:Class>
<owl:Class rdf:about="#PhoneNumber">
  <rdfs:subClassOf>
    <owl:Class rdf:about="#ID"/>
  </rdfs:subClassOf>
</owl:Class>
<owl:Class rdf:ID="UserID">
  <rdfs:subClassOf>
    <owl:Class rdf:about="#ID"/>
  </rdfs:subClassOf>
</owl:Class>
<owl:Class rdf:ID="TheaterName">
  <rdfs:subClassOf rdf:resource="#Text"/>
</owl:Class>
<owl:Class rdf:ID="MobileID">
  <rdfs:subClassOf>
    <owl:Class rdf:ID="MmsAddress"/>
  </rdfs:subClassOf>
  <rdfs:subClassOf rdf:resource="#SmsAddress"/>
</owl:Class>
<owl:Class rdf:ID="MsnAddress">
  <rdfs:subClassOf>
```

IA configuration files

```
<owl:Class rdf:about="#ID"/>
</rdfs:subClassOf>
</owl:Class>
<owl:Class rdf:ID="PriceInfo">
  <rdfs:subClassOf rdf:resource="#Text"/>
</owl:Class>
<owl:Class rdf:ID="RSSFeed">
  <rdfs:subClassOf rdf:resource="#Media_Content"/>
</owl:Class>
<owl:Class rdf:ID="WebRanking">
  <rdfs:subClassOf rdf:resource="#IO"/>
</owl:Class>
<owl:Class rdf:ID="AimAddress">
  <rdfs:subClassOf>
    <owl:Class rdf:about="#ID"/>
  </rdfs:subClassOf>
</owl:Class>
<owl:Class rdf:ID="FinancialInfo">
  <rdfs:subClassOf rdf:resource="#Text"/>
</owl:Class>
<owl:Class rdf:ID="BusinessTree">
  <rdfs:subClassOf rdf:resource="#IO"/>
</owl:Class>
<owl:Class rdf:ID="RestaurantInfoNorwegian">
  <rdfs:subClassOf rdf:resource="#NorwegianText"/>
</owl:Class>
<owl:Class rdf:ID="Price">
  <rdfs:subClassOf rdf:resource="#IO"/>
</owl:Class>
<owl:Class rdf:ID="RestaurantAddress">
  <rdfs:subClassOf rdf:resource="#Address"/>
</owl:Class>
<owl:Class rdf:ID="GeographicalInfo">
  <rdfs:subClassOf rdf:resource="#Text"/>
</owl:Class>
<owl:Class rdf:ID="CompanyName">
  <rdfs:subClassOf rdf:resource="#IO"/>
</owl:Class>
<owl:Class rdf:ID="Video">
  <rdfs:subClassOf rdf:resource="#Media_Content"/>
</owl:Class>
<owl:Class rdf:about="#MmsAddress">
  <rdfs:subClassOf>
    <owl:Class rdf:about="#ID"/>
  </rdfs:subClassOf>
</owl:Class>
<owl:Class rdf:ID="MP3">
  <rdfs:subClassOf rdf:resource="#Media_Content"/>
</owl:Class>
<owl:Class rdf:about="#ID">
  <rdfs:subClassOf rdf:resource="#IO"/>
</owl:Class>
<owl:Class rdf:ID="USPhoneNumber">
  <rdfs:subClassOf rdf:resource="#PhoneNumber"/>
</owl:Class>
<owl:Class rdf:ID="IcqAddress">
  <rdfs:subClassOf rdf:resource="#ID"/>
</owl:Class>
<owl:Class rdf:ID="CityInEngland">
  <rdfs:subClassOf>
    <owl:Class rdf:about="#CityName"/>
  </rdfs:subClassOf>
</owl:Class>
<owl:Class rdf:ID="DepartureTime">
  <rdfs:subClassOf rdf:resource="#Time"/>
</owl:Class>
<owl:Class rdf:ID="CellID">
```

IA configuration files

```
<rdfs:subClassOf rdf:resource="#Location"/>
</owl:Class>
<owl:Class rdf:ID="WCESStock">
  <rdfs:subClassOf>
    <owl:Class rdf:about="#StockName"/>
  </rdfs:subClassOf>
</owl:Class>
<owl:Class rdf:ID="Recipe">
  <rdfs:subClassOf rdf:resource="#Text"/>
</owl:Class>
<owl:Class rdf:ID="BusinessInfo">
  <rdfs:subClassOf rdf:resource="#Text"/>
</owl:Class>
<owl:Class rdf:about="#StockName">
  <rdfs:subClassOf rdf:resource="#IO"/>
</owl:Class>
<owl:Class rdf:ID="ResidenceInfo">
  <rdfs:subClassOf rdf:resource="#Text"/>
</owl:Class>
<owl:Class rdf:ID="AirportName">
  <rdfs:subClassOf rdf:resource="#IO"/>
</owl:Class>
<owl:Class rdf:ID="URI">
  <rdfs:subClassOf rdf:resource="#ID"/>
</owl:Class>
<owl:Class rdf:ID="COMEXStock">
  <rdfs:subClassOf rdf:resource="#StockName"/>
</owl:Class>
<owl:Class rdf:ID="CellNumber">
  <rdfs:subClassOf rdf:resource="#Location"/>
</owl:Class>
<owl:Class rdf:ID="Quote">
  <rdfs:subClassOf rdf:resource="#IO"/>
</owl:Class>
<owl:Class rdf:ID="StateName">
  <rdfs:subClassOf rdf:resource="#IO"/>
</owl:Class>
<owl:Class rdf:ID="JobInfo">
  <rdfs:subClassOf rdf:resource="#Text"/>
</owl:Class>
<owl:Class rdf:ID="GPSCoordinates">
  <rdfs:subClassOf rdf:resource="#Location"/>
</owl:Class>
<owl:Class rdf:ID="USStreetName">
  <rdfs:subClassOf rdf:resource="#StreetName"/>
</owl:Class>
<owl:Class rdf:ID="News">
  <rdfs:subClassOf rdf:resource="#Text"/>
</owl:Class>
<owl:Class rdf:ID="Temperature">
  <rdfs:subClassOf rdf:resource="#IO"/>
</owl:Class>
<owl:Class rdf:about="#CityName">
  <rdfs:subClassOf rdf:resource="#IO"/>
</owl:Class>
<owl:Class rdf:ID="IPAddress">
  <rdfs:subClassOf rdf:resource="#ID"/>
</owl:Class>
<owl:Class rdf:ID="EmailAddress">
  <rdfs:subClassOf rdf:resource="#ID"/>
</owl:Class>
<owl:Class rdf:ID="RestaurantLocation">
  <rdfs:subClassOf rdf:resource="#Address"/>
</owl:Class>
<owl:Class rdf:ID="RestaurantType">
  <rdfs:subClassOf rdf:resource="#IO"/>
</owl:Class>
```

IA configuration files

```
<owl:Class rdf:ID="DepartureDate">
  <rdfs:subClassOf rdf:resource="#Time"/>
</owl:Class>
<owl:Class>
<owl:Class rdf:ID="SpanishText">
  <rdfs:subClassOf rdf:resource="#Text"/>
</owl:Class>
<owl:Class rdf:ID="Longitude">
  <rdfs:subClassOf rdf:resource="#Location"/>
</owl:Class>
<CityName rdf:ID="Cardiff">
  <rdf:type rdf:resource="#CityInWales"/>
</CityName>
<CityName rdf:ID="Paris">
  <rdf:type rdf:resource="#CityInFrance"/>
  <rdf:type rdf:resource="#DepartCity"/>
  <rdf:type rdf:resource="#ArriveCity"/>
</CityName>
<CountryName rdf:ID="Norway"/>
<CityName rdf:ID="London">
  <rdf:type rdf:resource="#CityInBritish"/>
  <rdf:type rdf:resource="#DepartCity"/>
  <rdf:type rdf:resource="#ArriveCity"/>
</CityName>
<CityName rdf:ID="Edinburgh">
  <rdf:type rdf:resource="#CityInScotland"/>
</CityName>
<IPAddress rdf:ID="my_ipaddress"/>
<IPAddress rdf:ID="me_ipaddress"/>
<CityName rdf:ID="Bogota">
  <rdf:type rdf:resource="#CityInColumbia"/>
</CityName>
<CountryName rdf:ID="Finland"/>
<Location rdf:ID="my_place">
  <rdf:type rdf:resource="#UKAddress"/>
  <rdf:type rdf:resource="#CellID"/>
</Location>
<SmsAddress rdf:ID="my_telephone">
  <rdf:type rdf:resource="#PhoneNumber"/>
  <rdf:type rdf:resource="#MmsAddress"/>
</SmsAddress>
<CountryName rdf:ID="Estonia"/>
<USStreetName rdf:ID="Redmond"/>
<SmsAddress rdf:ID="my_phone">
  <rdf:type rdf:resource="#PhoneNumber"/>
  <rdf:type rdf:resource="#MmsAddress"/>
</SmsAddress>
<SmsAddress rdf:ID="my_mobile_phone">
  <rdf:type rdf:resource="#PhoneNumber"/>
  <rdf:type rdf:resource="#MmsAddress"/>
</SmsAddress>
<RestaurantType rdf:ID="japanese"/>
<IcqAddress rdf:ID="my_icq"/>
<Location rdf:ID="my_location">
  <rdf:type rdf:resource="#UKAddress"/>
  <rdf:type rdf:resource="#CellID"/>
</Location>
<RestaurantType rdf:ID="italian"/>
<CountryName rdf:ID="Sweden"/>
<RestaurantType rdf:ID="vegetarian"/>
<CityName rdf:ID="NewYork">
  <rdf:type rdf:resource="#CityInUnitedStates"/>
</CityName>
<Location rdf:ID="my_area">
  <rdf:type rdf:resource="#UKAddress"/>
  <rdf:type rdf:resource="#CellID"/>
</Location>
<EmailAddress rdf:ID="my_email"/>
```

IA configuration files

```
<EmailAddress rdf:ID="me_email"/>
<CountryName rdf:ID="Denmark"/>
<CityName rdf:ID="Toronto">
  <rdf:type rdf:resource="#CityInCanada"/>
</CityName>
<IPAddress rdf:ID="my_ip"/>
<CountryName rdf:ID="Cuba"/>
<CityName rdf:ID="Portsmouth">
  <rdf:type rdf:resource="#CityInEngland"/>
</CityName>
<CityName rdf:ID="Dublin">
  <rdf:type rdf:resource="#CityInNorthIreland"/>
</CityName>
</rdf:RDF>
<!-- Created with Protege (with OWL Plugin 3.3.1, Build 430) http://protege.stanford.edu -->
```

C.4 Synonym Ontology (Synonym.owl)

```
<?xml version="1.0"?>
<rdf:RDF
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns="http://www.spicerepository.net/Synonym.owl#"
  xmlns:owl="http://www.w3.org/2002/07/owl#"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema#"
  xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
  xml:base="http://www.spicerepository.net/Synonym.owl">
  <owl:Ontology rdf:about=""/>
  <owl:Class rdf:ID="Runmage">
    <rdfs:subClassOf>
      <owl:Class rdf:ID="Find"/>
    </rdfs:subClassOf>
  </owl:Class>
  <owl:Class rdf:ID="Movie">
    <rdfs:subClassOf>
      <owl:Class rdf:ID="Theater"/>
    </rdfs:subClassOf>
  </owl:Class>
  <owl:Class rdf:ID="Send">
    <rdfs:subClassOf rdf:resource="http://www.w3.org/2002/07/owl#Thing"/>
  </owl:Class>
  <owl:Class rdf:ID="Place">
    <rdfs:subClassOf>
      <owl:Class rdf:ID="Location"/>
    </rdfs:subClassOf>
  </owl:Class>
  <owl:Class rdf:ID="Reserve">
    <rdfs:subClassOf>
      <owl:Class rdf:ID="Book"/>
    </rdfs:subClassOf>
  </owl:Class>
  <owl:Class rdf:ID="Examine">
    <rdfs:subClassOf>
      <owl:Class rdf:about="#Find"/>
    </rdfs:subClassOf>
  </owl:Class>
  <owl:Class rdf:ID="Cinema">
    <rdfs:subClassOf rdf:resource="#Theater"/>
  </owl:Class>
  <owl:Class rdf:ID="Get">
    <rdfs:subClassOf rdf:resource="http://www.w3.org/2002/07/owl#Thing"/>
    <owl:equivalentClass>
      <owl:Class rdf:about="#Find"/>
    </owl:equivalentClass>
    <owl:equivalentClass rdf:resource="#Send"/>
  </owl:Class>
  <owl:Class rdf:ID="Search">
```

IA configuration files

```

<owl:equivalentClass>
  <owl:Class rdf:about="#Find"/>
</owl:equivalentClass>
<rdfs:subClassOf rdf:resource="http://www.w3.org/2002/07/owl#Thing"/>
</owl:Class>
<owl:Class rdf:ID="Engage">
  <rdfs:subClassOf rdf:resource="#Book"/>
</owl:Class>
<owl:Class rdf:ID="Hunt">
  <rdfs:subClassOf>
    <owl:Class rdf:about="#Find"/>
  </rdfs:subClassOf>
</owl:Class>
<owl:Class rdf:ID="film">
  <rdfs:subClassOf rdf:resource="#Theater"/>
</owl:Class>
<owl:Class rdf:about="#Find">
  <owl:equivalentClass rdf:resource="#Get"/>
  <owl:equivalentClass rdf:resource="#Search"/>
  <rdfs:subClassOf rdf:resource="http://www.w3.org/2002/07/owl#Thing"/>
</owl:Class>
<owl:Class rdf:ID="Explore">
  <rdfs:subClassOf rdf:resource="#Find"/>
</owl:Class>
<owl:Class rdf:ID="Seek">
  <rdfs:subClassOf rdf:resource="#Find"/>
</owl:Class>
<owl:Class rdf:ID="Investigate">
  <rdfs:subClassOf rdf:resource="#Find"/>
</owl:Class>
<owl:Class rdf:ID="Rifle">
  <rdfs:subClassOf rdf:resource="#Find"/>
</owl:Class>
<owl:Class rdf:ID="Order">
  <rdfs:subClassOf rdf:resource="#Book"/>
</owl:Class>
</rdf:RDF>

```

<!-- Created with Protege (with OWL Plugin 3.3.1, Build 430) <http://protege.stanford.edu> -->

C.5 Recommended Service Rule (ContextRule.txt)

```

! context rule, invoke operation of service when the context occurs
!
! Context      Service      operation      input parameter
!-----
"A2.05@telin.nl" "CITYTrafficReport" "ReportTrafficCondition" "Paris"
"A2.05@telin.nl" "MessagingServer" "SmsSender" "TrafficReport,00475612605"
"Canteen@telin.nl" "GlobalNews" "SendNews" "00475612605"
"street@telin.nl" "CITYTrafficReport" "ReportTrafficCondition" "Paris"
"street@telin.nl" "MessagingServer" "SmsSender" "TrafficReport,00475612605"
"restaurant@telin.nl" "GlobalWeatherForecast" "WeatherReport" "Paris"
"restaurant@telin.nl" "MessagingServer" "SmsSender" "WeatherReport,00475612605"
"customer_siteA@telin.nl" "DistanceService" "GetDistance" "cus-
tomer_siteA@telin.nl,customer_siteB@telin.nl"
"customer_siteA@telin.nl" "MessagingServer" "SmsSender" "Distance,00475612605"
"customer_siteB@telin.nl" "DistanceService" "GetDistance" "cus-
tomer_siteB@telin.nl,customer_siteC@telin.nl"
"customer_siteB@telin.nl" "MessagingServer" "SmsSender" "Distance,00475612605"
"customer_siteC@telin.nl" "DistanceService" "GetDistance" "cus-
tomer_siteC@telin.nl,customer_siteD@telin.nl"
"customer_siteC@telin.nl" "MessagingServer" "SmsSender" "Distance,00475612605"
"customer_siteD@telin.nl" "DistanceService" "GetDistance" "cus-
tomer_siteD@telin.nl,customer_siteE@telin.nl"
"customer_siteD@telin.nl" "MessagingServer" "SmsSender" "Distance,00475612605"
"customer_siteE@telin.nl" "MessagingServer" "SMTPServer" "deal.doc,office@telin.nl"

```

C.6 Restriction Service Rule (RestrictionRule.txt)

```
! restriction rule, restrict an access to service when the context occurs
!
! Context      Service
!-----
"A2.07@telin.nl"  "MailDaemon,EmailDelivery"
```

C.7 Dictionary file (norwegian2english.txt)

```
! convert norwegian vocab to english vocab
!
! Norwegian      English
!-----
"finne"          "find"
"addressee"     "addressee"
"pronoun"       "pronoun"
"trafikkforhold" "traffic"
"conjunction"   "and"
"conjoined"     "and"
"sende"         "send"
"pr."           "by"
"email"         "email"
"i"             "in"
"omrade"        "area"
"vegetarrestaurant" "vegetarian restaurant"
"poss"          "own"
"disjunction"   "or"
"sms"           "sms"
"værrapport"    "weather"
"oversette"     "translate"
"til"           "to"
"engelsk"       "english"
"coincide"      "and"
"adresse"       "address"
"teater"        "theater"
"flight"        "flight"
"fra"           "from"
"for"           "for"
"soke"          "search"
"bestille"     "book"
```

C.8 MRS for “Find traffic in Paris and send them by email” (nlp.xml)

```
<mrs>
<label vid='1'/><var vid='2'/>
<ep cfrom='0' cto='4'><spred>_finne_v_rel</spred><label vid='3'/>
<fvpair><rargname>ARG0</rargname><var vid='2' sort='e'>
<extrapair><path>SF</path><value>COMM</value></extrapair>
<extrapair><path>E.MOOD</path><value>IMPERATIVE</value></extrapair>
<extrapair><path>E.TENSE</path><value>TENSE</value></extrapair>
<extrapair><path>E.ASPECT</path><value>SEMSORT</value></extrapair>
<extrapair><path>E.DELIMITED</path><value>+</value></extrapair>
<extrapair><path>PATH-TELIC</path><value>BOOL</value></extrapair>
<extrapair><path>SIT-TYPE</path><value>SEMSORT</value></extrapair>
<extrapair><path>DISC-MOVE</path><value>DISCMODE</value></extrapair>
<extrapair><path>WH</path><value>BOOL</value></extrapair>
<extrapair><path>ROLE</path><value>ROLE</value></extrapair></var></fvpair>
<fvpair><rargname>ARG1</rargname><var vid='5' sort='x'>
<extrapair><path>WH</path><value>BOOL</value></extrapair>
<extrapair><path>ROLE</path><value>ROLE</value></extrapair>
<extrapair><path>PNG.NG.NUM</path><value>NUM</value></extrapair>
<extrapair><path>PNG.NG.GEN</path><value>M</value></extrapair>
<extrapair><path>PNG.PERS</path><value>SECPERS</value></extrapair></var></fvpair>
<fvpair><rargname>ARG2</rargname><var vid='4' sort='x'>
```



```

<extrapair><path>PNG.NG.GEN</path><value>N</value></extrapair>
<extrapair><path>PNG.PERS</path><value>THIRDPERS</value></extrapair>
<extrapair><path>WH</path><value>-</value></extrapair>
<extrapair><path>ROLE</path><value>GLBTYPE49</value></extrapair>
<extrapair><path>BOUNDED</path><value>BOOL</value></extrapair></var></fvpair></ep>
<hcons hreln='req'><hi><var vid='8' sort='h'></var></hi><lo><var vid='6' sort='h'></var></lo></hcons>
<hcons hreln='req'><hi><var vid='12' sort='h'></var></hi><lo><var vid='10' sort='h'></var></lo></hcons>
<hcons hreln='req'><hi><var vid='19' sort='h'></var></hi><lo><var vid='17' sort='h'></var></lo></hcons>
<hcons hreln='req'><hi><var vid='25' sort='h'></var></hi><lo><var vid='21' sort='h'></var></lo></hcons>
<hcons hreln='req'><hi><var vid='31' sort='h'></var></hi><lo><var vid='29' sort='h'></var></lo></hcons>
<hcons hreln='req'><hi><var vid='35' sort='h'></var></hi><lo><var vid='33' sort='h'></var></lo></hcons>
<hcons hreln='req'><hi><var vid='42' sort='h'></var></hi><lo><var vid='40' sort='h'></var></lo></hcons>
</mrs>

```

C.9 MRS for “Find vegetarian restaurant in my area and send it by sms or email”(nlp2.xml)

```

<mrs>
<label vid='1'><var vid='2'>
<ep cfrom='0' cto='4'><spred>_finne_v_rel</spred><label vid='3'>
<fvpair><rargname>ARG0</rargname><var vid='2' sort='e'>
<extrapair><path>SF</path><value>COMM</value></extrapair>
<extrapair><path>E.MOOD</path><value>IMPERATIVE</value></extrapair>
<extrapair><path>E.TENSE</path><value>TENSE</value></extrapair>
<extrapair><path>E.ASPECT</path><value>SEMSORT</value></extrapair>
<extrapair><path>E.DELIMITED</path><value>BOOL</value></extrapair>
<extrapair><path>PATH.TELIC</path><value>BOOL</value></extrapair>
<extrapair><path>SIT-TYPE</path><value>SEMSORT</value></extrapair>
<extrapair><path>DISC-MOVE</path><value>DISCMODE</value></extrapair>
<extrapair><path>WH</path><value>BOOL</value></extrapair>
<extrapair><path>ROLE</path><value>ROLE</value></extrapair></var></fvpair>
<fvpair><rargname>ARG1</rargname><var vid='5' sort='x'>
<extrapair><path>WH</path><value>BOOL</value></extrapair>
<extrapair><path>ROLE</path><value>ROLE</value></extrapair>
<extrapair><path>PNG.NG.NUM</path><value>NUM</value></extrapair>
<extrapair><path>PNG.NG.GEN</path><value>M</value></extrapair>
<extrapair><path>PNG.PERS</path><value>SECPERS</value></extrapair></var></fvpair>
<fvpair><rargname>ARG2</rargname><var vid='4' sort='x'>
<extrapair><path>ROLE</path><value>XDIM-NONINIT</value></extrapair>
<extrapair><path>PNG.NG.NUM</path><value>SING</value></extrapair>
<extrapair><path>PNG.NG.GEN</path><value>M</value></extrapair>
<extrapair><path>PNG.PERS</path><value>THIRDPERS</value></extrapair>
<extrapair><path>WH</path><value>-</value></extrapair>
<extrapair><path>BOUNDED</path><value>BOOL</value></extrapair></var></fvpair></ep>
<ep cfrom='0' cto='4'><spred>addressee-rel</spred><label vid='6'>
<fvpair><rargname>ARG0</rargname><var vid='5' sort='x'>
<extrapair><path>WH</path><value>BOOL</value></extrapair>
<extrapair><path>ROLE</path><value>ROLE</value></extrapair>
<extrapair><path>PNG.NG.NUM</path><value>NUM</value></extrapair>
<extrapair><path>PNG.NG.GEN</path><value>M</value></extrapair>
<extrapair><path>PNG.PERS</path><value>SECPERS</value></extrapair></var></fvpair></ep>
<ep cfrom='0' cto='4'><spred>_pronoun_q_rel</spred><label vid='7'>
<fvpair><rargname>ARG0</rargname><var vid='5' sort='x'>
<extrapair><path>WH</path><value>BOOL</value></extrapair>
<extrapair><path>ROLE</path><value>ROLE</value></extrapair>
<extrapair><path>PNG.NG.NUM</path><value>NUM</value></extrapair>
<extrapair><path>PNG.NG.GEN</path><value>M</value></extrapair>
<extrapair><path>PNG.PERS</path><value>SECPERS</value></extrapair></var></fvpair>
<fvpair><rargname>RSTR</rargname><var vid='8' sort='h'>
<extrapair><path>ROLE</path><value>ROLE</value></extrapair></var></fvpair>
<fvpair><rargname>BODY</rargname><var vid='9' sort='h'>
<extrapair><path>ROLE</path><value>ROLE</value></extrapair></var></fvpair></ep>
<ep cfrom='5' cto='22'><spred>_vegetarrestaurant_n_rel</spred><label vid='10'>
<fvpair><rargname>ARG0</rargname><var vid='4' sort='x'>
<extrapair><path>ROLE</path><value>XDIM-NONINIT</value></extrapair>

```



```

<hcons hreln='req'><hi><var vid='25' sort='h'></var></hi><lo><var vid='22' sort='h'></var></lo></hcons>
<hcons hreln='req'><hi><var vid='31' sort='h'></var></hi><lo><var vid='27' sort='h'></var></lo></hcons>
<hcons hreln='req'><hi><var vid='37' sort='h'></var></hi><lo><var vid='35' sort='h'></var></lo></hcons>
<hcons hreln='req'><hi><var vid='41' sort='h'></var></hi><lo><var vid='39' sort='h'></var></lo></hcons>
<hcons hreln='req'><hi><var vid='49' sort='h'></var></hi><lo><var vid='46' sort='h'></var></lo></hcons>
<hcons hreln='req'><hi><var vid='56' sort='h'></var></hi><lo><var vid='51' sort='h'></var></lo></hcons>
<hcons hreln='req'><hi><var vid='59' sort='h'></var></hi><lo><var vid='53' sort='h'></var></lo></hcons>
</mrs>

```

C.10 MRS for “Find weather report in Paris, translated to English and send it by email”(nlp3.xml)

```

<mrs>
<label vid='1'><var vid='2'>
<ep cfrom='0' cto='4'><spred>_finne_v_rel</spred><label vid='3'>
<fvpair><rargname>ARG0</rargname><var vid='2' sort='e'>
<extrapair><path>SF</path><value>COMM</value></extrapair>
<extrapair><path>E.MOOD</path><value>IMPERATIVE</value></extrapair>
<extrapair><path>E.TENSE</path><value>TENSE</value></extrapair>
<extrapair><path>E.ASPECT</path><value>SEMSORT</value></extrapair>
<extrapair><path>E.DELIMITED</path><value>BOOL</value></extrapair>
<extrapair><path>PATH.TELIC</path><value>BOOL</value></extrapair>
<extrapair><path>SIT-TYPE</path><value>SEMSORT</value></extrapair>
<extrapair><path>DISC-MOVE</path><value>DISCMODE</value></extrapair>
<extrapair><path>WH</path><value>BOOL</value></extrapair>
<extrapair><path>ROLE</path><value>ROLE</value></extrapair></var></fvpair>
<fvpair><rargname>ARG1</rargname><var vid='5' sort='x'>
<extrapair><path>WH</path><value>BOOL</value></extrapair>
<extrapair><path>ROLE</path><value>ROLE</value></extrapair>
<extrapair><path>PNG.NG.NUM</path><value>NUM</value></extrapair>
<extrapair><path>PNG.NG.GEN</path><value>M</value></extrapair>
<extrapair><path>PNG.PERS</path><value>SECPERS</value></extrapair></var></fvpair>
<fvpair><rargname>ARG2</rargname><var vid='4' sort='x'>
<extrapair><path>ROLE</path><value>GLBTYP47</value></extrapair>
<extrapair><path>PNG.NG.NUM</path><value>SING</value></extrapair>
<extrapair><path>PNG.NG.GEN</path><value>M</value></extrapair>
<extrapair><path>PNG.PERS</path><value>THIRDPERS</value></extrapair>
<extrapair><path>WH</path><value>-</value></extrapair>
<extrapair><path>BOUNDED</path><value>BOOL</value></extrapair></var></fvpair></ep>
<ep cfrom='0' cto='4'><spred>addressee-rel</spred><label vid='6'>
<fvpair><rargname>ARG0</rargname><var vid='5' sort='x'>
<extrapair><path>WH</path><value>BOOL</value></extrapair>
<extrapair><path>ROLE</path><value>ROLE</value></extrapair>
<extrapair><path>PNG.NG.NUM</path><value>NUM</value></extrapair>
<extrapair><path>PNG.NG.GEN</path><value>M</value></extrapair>
<extrapair><path>PNG.PERS</path><value>SECPERS</value></extrapair></var></fvpair></ep>
<ep cfrom='0' cto='4'><spred>_pronoun_q_rel</spred><label vid='7'>
<fvpair><rargname>ARG0</rargname><var vid='5' sort='x'>
<extrapair><path>WH</path><value>BOOL</value></extrapair>
<extrapair><path>ROLE</path><value>ROLE</value></extrapair>
<extrapair><path>PNG.NG.NUM</path><value>NUM</value></extrapair>
<extrapair><path>PNG.NG.GEN</path><value>M</value></extrapair>
<extrapair><path>PNG.PERS</path><value>SECPERS</value></extrapair></var></fvpair>
<fvpair><rargname>RSTR</rargname><var vid='8' sort='h'>
<extrapair><path>ROLE</path><value>ROLE</value></extrapair></var></fvpair>
<fvpair><rargname>BODY</rargname><var vid='9' sort='h'>
<extrapair><path>ROLE</path><value>ROLE</value></extrapair></var></fvpair></ep>
<ep cfrom='5' cto='15'><spred>_vÃrreport_n_rel</spred><label vid='10'>
<fvpair><rargname>ARG0</rargname><var vid='4' sort='x'>
<extrapair><path>ROLE</path><value>GLBTYP47</value></extrapair>
<extrapair><path>PNG.NG.NUM</path><value>SING</value></extrapair>
<extrapair><path>PNG.NG.GEN</path><value>M</value></extrapair>
<extrapair><path>PNG.PERS</path><value>THIRDPERS</value></extrapair>
<extrapair><path>WH</path><value>-</value></extrapair>
<extrapair><path>BOUNDED</path><value>BOOL</value></extrapair></var></fvpair></ep>

```


C.11 MRS for “Search a flight from London to Paris on 10-06-2008 and book it”(nlp5.xml)

```

<mrs>
<label vid='1'/><var vid='2'/>
<ep cfrom='0' cto='3'><spread>_soke_v_rel</spread><label vid='3'/>
<fvpair><rargname>ARG0</rargname><var vid='2' sort='e'>
<extrapair><path>WH</path><value>BOOL</value></extrapair>
<extrapair><path>ROLE</path><value>ROLE</value></extrapair>
<extrapair><path>SF</path><value>COMM</value></extrapair>
<extrapair><path>E.MOOD</path><value>IMPERATIVE</value></extrapair>
<extrapair><path>E.TENSE</path><value>TENSE</value></extrapair>
<extrapair><path>E.ASPECT</path><value>SEMSORT</value></extrapair>
<extrapair><path>E.DELIMITED</path><value>BOOL</value></extrapair>
<extrapair><path>PATH-TELIC</path><value>BOOL</value></extrapair>
<extrapair><path>SIT-TYPE</path><value>SEMSORT</value></extrapair>
<extrapair><path>DISC-MOVE</path><value>DISCMODE</value></extrapair></var></fvpair>
<fvpair><rargname>ARG1</rargname><var vid='5' sort='x'>
<extrapair><path>WH</path><value>BOOL</value></extrapair>
<extrapair><path>ROLE</path><value>ROLE</value></extrapair>
<extrapair><path>PNG.NG.NUM</path><value>NUM</value></extrapair>
<extrapair><path>PNG.NG.GEN</path><value>M</value></extrapair>
<extrapair><path>PNG.PERS</path><value>SECPERS</value></extrapair></var></fvpair>
<fvpair><rargname>ARG2</rargname><var vid='4' sort='x'>
<extrapair><path>ROLE</path><value>GLBTYPE47</value></extrapair>
<extrapair><path>PNG.NG.NUM</path><value>SING</value></extrapair>
<extrapair><path>PNG.NG.GEN</path><value>M</value></extrapair>
<extrapair><path>PNG.PERS</path><value>THIRDPERS</value></extrapair>
<extrapair><path>WH</path><value></value></extrapair>
<extrapair><path>BOUNDED</path><value>BOOL</value></extrapair></var></fvpair></ep>
<ep cfrom='0' cto='3'><spread>addressee-rel</spread><label vid='6'/>
<fvpair><rargname>ARG0</rargname><var vid='5' sort='x'>
<extrapair><path>WH</path><value>BOOL</value></extrapair>
<extrapair><path>ROLE</path><value>ROLE</value></extrapair>
<extrapair><path>PNG.NG.NUM</path><value>NUM</value></extrapair>
<extrapair><path>PNG.NG.GEN</path><value>M</value></extrapair>
<extrapair><path>PNG.PERS</path><value>SECPERS</value></extrapair></var></fvpair></ep>
<ep cfrom='0' cto='3'><spread>_pronoun_q_rel</spread><label vid='7'/>
<fvpair><rargname>ARG0</rargname><var vid='5' sort='x'>
<extrapair><path>WH</path><value>BOOL</value></extrapair>
<extrapair><path>ROLE</path><value>ROLE</value></extrapair>
<extrapair><path>PNG.NG.NUM</path><value>NUM</value></extrapair>
<extrapair><path>PNG.NG.GEN</path><value>M</value></extrapair>
<extrapair><path>PNG.PERS</path><value>SECPERS</value></extrapair></var></fvpair>
<fvpair><rargname>RSTR</rargname><var vid='8' sort='h'>
<extrapair><path>ROLE</path><value>ROLE</value></extrapair></var></fvpair>
<fvpair><rargname>BODY</rargname><var vid='9' sort='h'>
<extrapair><path>ROLE</path><value>ROLE</value></extrapair></var></fvpair></ep>
<ep cfrom='4' cto='6'><spread>exactly_1_q_rel</spread><label vid='10'/>
<fvpair><rargname>ARG0</rargname><var vid='4' sort='x'>
<extrapair><path>ROLE</path><value>GLBTYPE47</value></extrapair>
<extrapair><path>PNG.NG.NUM</path><value>SING</value></extrapair>
<extrapair><path>PNG.NG.GEN</path><value>M</value></extrapair>
<extrapair><path>PNG.PERS</path><value>THIRDPERS</value></extrapair>
<extrapair><path>WH</path><value></value></extrapair>
<extrapair><path>BOUNDED</path><value>BOOL</value></extrapair></var></fvpair>
<fvpair><rargname>RSTR</rargname><var vid='11' sort='h'>
<extrapair><path>ROLE</path><value>ROLE</value></extrapair></var></fvpair>
<fvpair><rargname>BODY</rargname><var vid='12' sort='h'>
<extrapair><path>ROLE</path><value>ROLE</value></extrapair></var></fvpair></ep>
<ep cfrom='7' cto='13'><spread>_flight_n_rel</spread><label vid='13'/>
<fvpair><rargname>ARG0</rargname><var vid='4' sort='x'>
<extrapair><path>ROLE</path><value>GLBTYPE47</value></extrapair>
<extrapair><path>PNG.NG.NUM</path><value>SING</value></extrapair>
<extrapair><path>PNG.NG.GEN</path><value>M</value></extrapair>

```



```

<extrapair><path>ROLE</path><value>ROLE</value></extrapair>
<extrapair><path>SF</path><value>COMM</value></extrapair>
<extrapair><path>E.MOOD</path><value>IMPERATIVE</value></extrapair>
<extrapair><path>E.TENSE</path><value>TENSE</value></extrapair>
<extrapair><path>E.ASPECT</path><value>SEMSORT</value></extrapair>
<extrapair><path>E.DELIMITED</path><value>BOOL</value></extrapair>
<extrapair><path>PATH-TELIC</path><value>BOOL</value></extrapair>
<extrapair><path>SIT-TYPE</path><value>SEMSORT</value></extrapair>
<extrapair><path>DISC-MOVE</path><value>DISCMODE</value></extrapair></var></fvpair></ep>
<hcons hreln='req'><hi><var vid='8' sort='h'></var></hi></lo><var vid='6' sort='h'></var></lo></hcons>
<hcons hreln='req'><hi><var vid='11' sort='h'></var></hi></lo><var vid='13' sort='h'></var></lo></hcons>
<hcons hreln='req'><hi><var vid='18' sort='h'></var></hi></lo><var vid='16' sort='h'></var></lo></hcons>
<hcons hreln='req'><hi><var vid='24' sort='h'></var></hi></lo><var vid='22' sort='h'></var></lo></hcons>
<hcons hreln='req'><hi><var vid='30' sort='h'></var></hi></lo><var vid='28' sort='h'></var></lo></hcons>
<hcons hreln='req'><hi><var vid='36' sort='h'></var></hi></lo><var vid='32' sort='h'></var></lo></hcons>
<hcons hreln='req'><hi><var vid='42' sort='h'></var></hi></lo><var vid='40' sort='h'></var></lo></hcons>
<hcons hreln='req'><hi><var vid='46' sort='h'></var></hi></lo><var vid='44' sort='h'></var></lo></hcons>
</mrs>

```

C.12 MRS for “Find an address of theater in my area and send it to my email”(nlp4.xml)

```

<mrs>
<label vid='1'><var vid='2'>
<ep cfrom='0' cto='4'><spread>_finne_v_rel</spread><label vid='3'>
<fvpair><rargname>ARG0</rargname><var vid='2' sort='e'>
<extrapair><path>WH</path><value>BOOL</value></extrapair>
<extrapair><path>ROLE</path><value>ROLE</value></extrapair>
<extrapair><path>SF</path><value>COMM</value></extrapair>
<extrapair><path>E.MOOD</path><value>IMPERATIVE</value></extrapair>
<extrapair><path>E.TENSE</path><value>TENSE</value></extrapair>
<extrapair><path>E.ASPECT</path><value>SEMSORT</value></extrapair>
<extrapair><path>E.DELIMITED</path><value>+</value></extrapair>
<extrapair><path>PATH-TELIC</path><value>BOOL</value></extrapair>
<extrapair><path>SIT-TYPE</path><value>SEMSORT</value></extrapair>
<extrapair><path>DISC-MOVE</path><value>DISCMODE</value></extrapair></var></fvpair>
<fvpair><rargname>ARG1</rargname><var vid='5' sort='x'>
<extrapair><path>WH</path><value>BOOL</value></extrapair>
<extrapair><path>ROLE</path><value>ROLE</value></extrapair>
<extrapair><path>PNG.NG.NUM</path><value>NUM</value></extrapair>
<extrapair><path>PNG.NG.GEN</path><value>M</value></extrapair>
<extrapair><path>PNG.PERS</path><value>SECPERS</value></extrapair></var></fvpair>
<fvpair><rargname>ARG2</rargname><var vid='4' sort='x'>
<extrapair><path>ROLE</path><value>NON-INITIATOR</value></extrapair>
<extrapair><path>PNG.NG.NUM</path><value>SING</value></extrapair>
<extrapair><path>PNG.NG.GEN</path><value>M</value></extrapair>
<extrapair><path>PNG.PERS</path><value>THIRDPERS</value></extrapair>
<extrapair><path>WH</path><value>-</value></extrapair>
<extrapair><path>BOUNDED</path><value>+</value></extrapair></var></fvpair></ep>
<ep cfrom='0' cto='4'><spread>addressee-rel</spread><label vid='6'>
<fvpair><rargname>ARG0</rargname><var vid='5' sort='x'>
<extrapair><path>WH</path><value>BOOL</value></extrapair>
<extrapair><path>ROLE</path><value>ROLE</value></extrapair>
<extrapair><path>PNG.NG.NUM</path><value>NUM</value></extrapair>
<extrapair><path>PNG.NG.GEN</path><value>M</value></extrapair>
<extrapair><path>PNG.PERS</path><value>SECPERS</value></extrapair></var></fvpair></ep>
<ep cfrom='0' cto='4'><spread>_pronoun_q_rel</spread><label vid='7'>
<fvpair><rargname>ARG0</rargname><var vid='5' sort='x'>
<extrapair><path>WH</path><value>BOOL</value></extrapair>
<extrapair><path>ROLE</path><value>ROLE</value></extrapair>
<extrapair><path>PNG.NG.NUM</path><value>NUM</value></extrapair>
<extrapair><path>PNG.NG.GEN</path><value>M</value></extrapair>
<extrapair><path>PNG.PERS</path><value>SECPERS</value></extrapair></var></fvpair>
<fvpair><rargname>RSTR</rargname><var vid='8' sort='h'>

```


IA configuration files

```
<extrapair><path>ROLE</path><value>ROLE</value></extrapair></var></fvpair>
<fvpair><rargname>ARG1</rargname><var vid='4' sort='x'>
<extrapair><path>ROLE</path><value>NON-INITIATOR</value></extrapair>
<extrapair><path>PNG.NG.NUM</path><value>SING</value></extrapair>
<extrapair><path>PNG.NG.GEN</path><value>M</value></extrapair>
<extrapair><path>PNG.PERS</path><value>THIRDPERS</value></extrapair>
<extrapair><path>WH</path><value>-</value></extrapair>
<extrapair><path>BOUNDED</path><value>+</value></extrapair></var></fvpair>
<fvpair><rargname>ARG2</rargname><var vid='38' sort='x'>
<extrapair><path>PNG.NG.NUM</path><value>SING</value></extrapair>
<extrapair><path>PNG.NG.GEN</path><value>M</value></extrapair>
<extrapair><path>PNG.PERS</path><value>THIRDPERS</value></extrapair>
<extrapair><path>WH</path><value>-</value></extrapair>
<extrapair><path>ROLE</path><value>GLBTYPE47</value></extrapair>
<extrapair><path>BOUNDED</path><value>BOOL</value></extrapair></var></fvpair></ep>
<ep cfrom='67' cto='68'><spread>_period-punctuated_rel</spread><label vid='3'>
<fvpair><rargname>ARG0</rargname><var vid='65' sort='u'>
<extrapair><path>WH</path><value>BOOL</value></extrapair>
<extrapair><path>ROLE</path><value>ROLE</value></extrapair></var></fvpair>
<fvpair><rargname>ARG1</rargname><var vid='2' sort='e'>
<extrapair><path>WH</path><value>BOOL</value></extrapair>
<extrapair><path>ROLE</path><value>ROLE</value></extrapair>
<extrapair><path>SF</path><value>COMM</value></extrapair>
<extrapair><path>E.MOOD</path><value>IMPERATIVE</value></extrapair>
<extrapair><path>E.TENSE</path><value>TENSE</value></extrapair>
<extrapair><path>E.ASPECT</path><value>SEMSORT</value></extrapair>
<extrapair><path>E.DELIMITED</path><value>+</value></extrapair>
<extrapair><path>PATH-TELIC</path><value>BOOL</value></extrapair>
<extrapair><path>SIT-TYPE</path><value>SEMSORT</value></extrapair>
<extrapair><path>DISC-MOVE</path><value>DISCMODE</value></extrapair></var></fvpair></ep>
<hcons hreln='req'><hi><var vid='8' sort='h'></var></hi><lo><var vid='6' sort='h'></var></lo></hcons>
<hcons hreln='req'><hi><var vid='12' sort='h'></var></hi><lo><var vid='10' sort='h'></var></lo></hcons>
<hcons hreln='req'><hi><var vid='17' sort='h'></var></hi><lo><var vid='19' sort='h'></var></lo></hcons>
<hcons hreln='req'><hi><var vid='25' sort='h'></var></hi><lo><var vid='22' sort='h'></var></lo></hcons>
<hcons hreln='req'><hi><var vid='30' sort='h'></var></hi><lo><var vid='27' sort='h'></var></lo></hcons>
<hcons hreln='req'><hi><var vid='36' sort='h'></var></hi><lo><var vid='32' sort='h'></var></lo></hcons>
<hcons hreln='req'><hi><var vid='42' sort='h'></var></hi><lo><var vid='40' sort='h'></var></lo></hcons>
<hcons hreln='req'><hi><var vid='46' sort='h'></var></hi><lo><var vid='44' sort='h'></var></lo></hcons>
<hcons hreln='req'><hi><var vid='54' sort='h'></var></hi><lo><var vid='51' sort='h'></var></lo></hcons>
<hcons hreln='req'><hi><var vid='59' sort='h'></var></hi><lo><var vid='56' sort='h'></var></lo></hcons>
</mrs>
```


Appendix D

User Manual

In this section, we describe how to install and utilize IA. The package IA can be installed as follows:

- 1) Open eclipse platform. Then, select File -> Import -> Existing Projects into Workspace
- 2) Select the package IA (in the CD attached with this thesis)
- 3) Select package IA, right click on the package name, select Properties
- 4) Select “Java Build Path”. Choose Libraries tab. Select “Add External JARs”
- 5) Select all jar files in the lib folder of package IA

It is worth noting that ,in order to test a context handling feature of IA, the CMF package has to be installed. To install CMF, we follow the same steps as IA. To run context source of CMF (that uses build script), we open the Ant view by selecting Window -> Show View -> Ant. Then, we select “compileAll” to compile, among other things, the context-source file. Finally, we select “runTutorial” to activate the context source. The windows of CMF will be populated as shown in Figure D-1.

We can configure the context source by selecting “ExampleUserLocation0”. Then, we press the “Config” button. The configuration windows will be populated as shown in Figure D-2. We can add the new location and the new user by filling in the “Spaces” and “Users” box, respectively. It is worth noting that we ignore the value of SimulationDelay because we control context source manually. Moreover, we ignore the value of Logfilename as, for our scenarios, we are not interested in the log file produced by context source. After the configuration has been completed, we press the “Ok” button.

To run the context source, we double-click on ExampleUserLocation0 in Figure D-1. The windows of the context source will be populated as shown in Figure D-3. To publish the user context, we select the user and his location from the drop-down list. Then, we press “Send Update” button. To terminate context source and CMF, we press the cross button in the top-right corner.

It is worth noting that the context source explained in this section is used to publish location context. To publish the security context, we have to modify the context source file of

nl.telin.cmf.tutorial (by commenting out part of the source code) as shown in Figure D-4. Then, we start up the context source again as explained in the preceding paragraphs.

To run IA, we select UI.java of package ui. Then, we select “Project” -> “Build Project” to compile all source files. After that, we select Run -> “Run As” -> “Java Application”. IA window should be populated as shown in Figure D-5.

To activate the input box, we press “new” button. We put a user request in the box next to “Natural Language Request” text. Then, we press “Save” button to activate the “Submit” button. Finally, we press “submit” button to send the request to IA. The result will be shown in “Result” area. The result consists of number of service compositions (# Composition) and details (Details) of the service compositions. The Output dialogue box informs us about the location where SPATEL files are saved.

To submit another request, we press “Delete” button to clear the result. Then, we repeat the same steps to send the request to IA as before. To enable context-handling feature, we select the Enable radio box of “Context Handling”. Similarly, to utilize MRS, we select the Enable radio box of “Natural Language”. Lastly, to peacefully terminate IA, we press the “Exit” button.

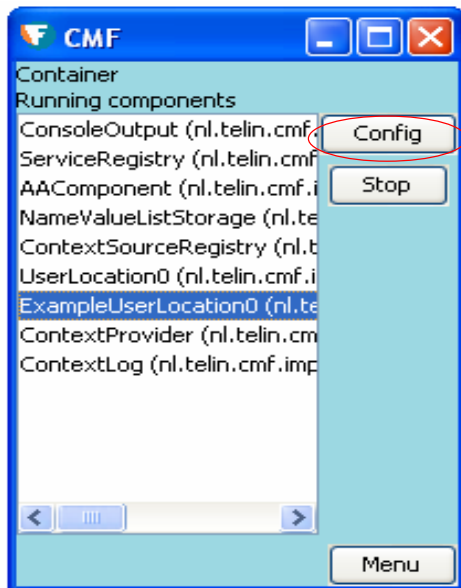


Figure D-1: CMF user interface

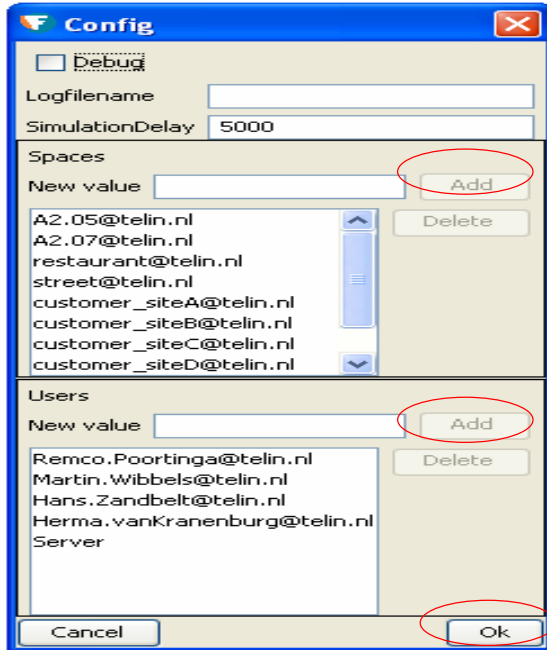


Figure D-2: CMF configuration window

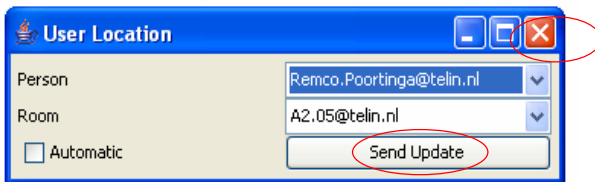


Figure D-3: Context source windows

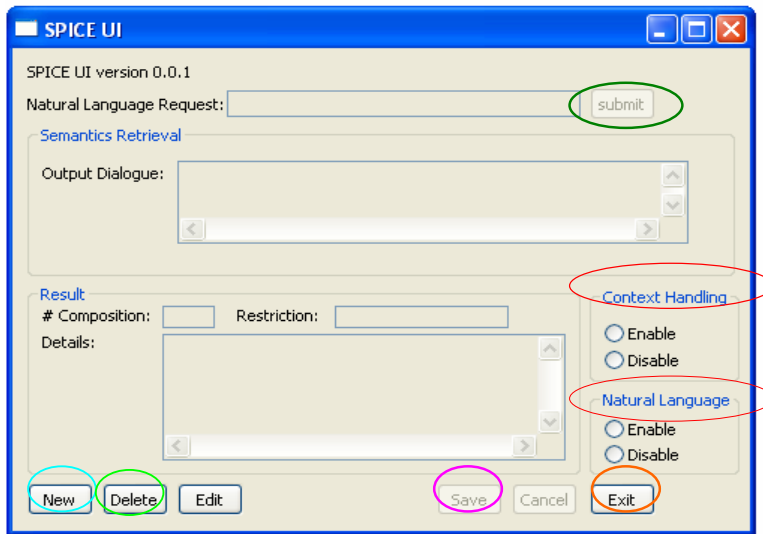


Figure D-5: IA user interface

Appendix E

Hardware and Software Specification

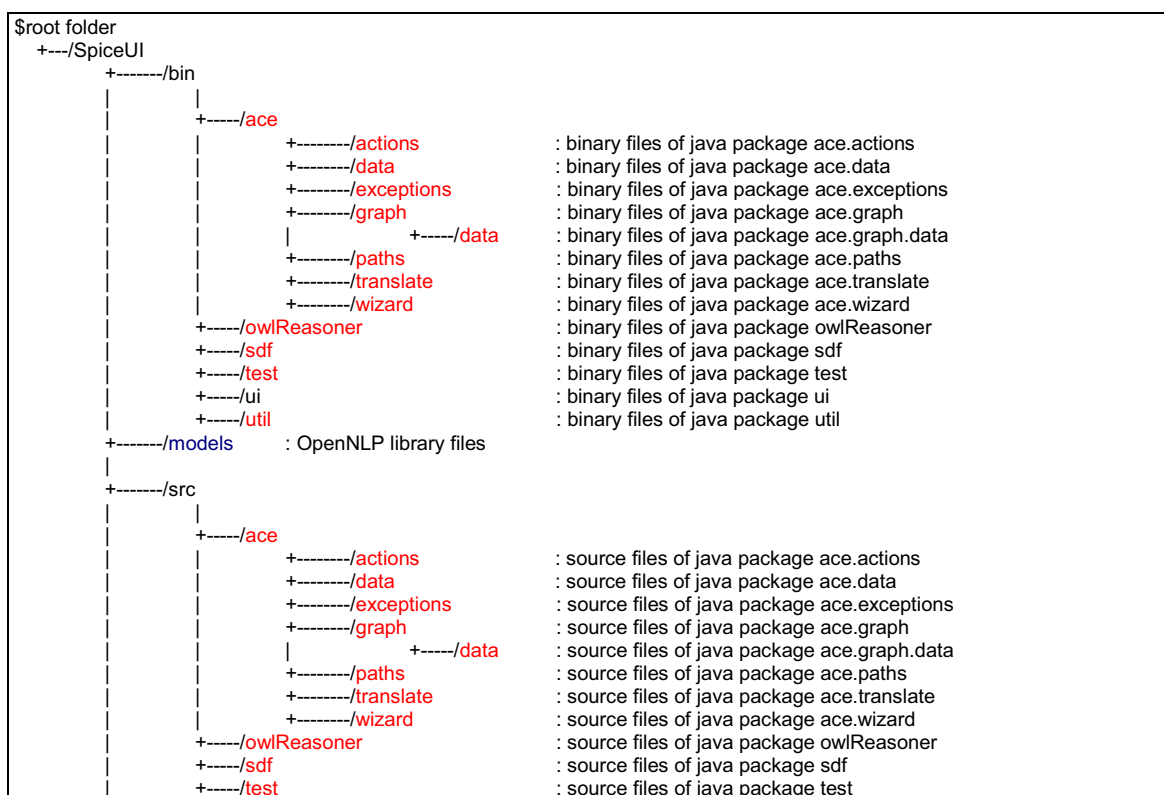
In this section, we list specifications of hardware and software that we used in this thesis.

<p>CPU: Intel(R) Pentium(R) 4 CPU 2.60 GHz RAM: 1 GB HDD: 9 GB OS: Microsoft Windows XP Professional Version 2002 Service Pack 2 Eclipse Platform: Version 3.3.1.1, Build id: M20071023-1652 Ontology Editor: Protégé 3.3.1 NLP Library: OpenNLP 1.3.0 GUI Library: SWT 3.3.2 Semantic web framework: Jena 2.5.5</p>

Appendix F

IA Directory Structure

Note that the packages that are in **red** have been obtained from another related project (ACE). The package that is in **blue** is a part of OpenNLP API. It has been obtained from the publisher site (<http://opennlp.sourceforge.net/>). The files that are in **green** were generated by a software (we called it as NLP-tool in this thesis) provided by professor Lars Hellan at Department of Linguistics, Norwegian University of Science and Technology (NTNU) The files and packages that are not in colour have been developed in this thesis.



IA Directory Structure

	+----/ui	: source files of java package ui
	+----/util	: source files of java package util
	+----/lib	: library files of IA
	+----.classpath	: eclipse environmental file
	+----.project	: eclipse environmental file
	+----ContextRule.txt	: recommended service rule file
	+----Synonym.owl	: synonym ontology file
	+----norwegian2english.txt	: dictionary file
	+----/spatel	: spatel output folder (for user request) and the test files
	+----Goals.owl	: goal ontology file
	+----IO.owl	: input/output ontology file
	+----services.csv	: service repository file
	+----RestrictionRule.txt	: restriction service rule file
	+----/spatel2	: spatel output folder (for context-handling) and the test files
	+----nlp.xml	: MRS file for IA correctness test
	+----nlp2.xml	: MRS file for scenario1
	+----nlp3.xml	: MRS file for scenario 2-1
	+----nlp5.xml	: MRS file for scenario 2-2
	+----nlp4.xml	: MRS file for scenario 3

Appendix G

StopWatch.java

Note that the source code is originally defined in

<http://www.javapractices.com/topic/TopicAction.do?Id=85>.

```
package ui;

/**
 *
 * Allows timing of the execution of any block of code.
 */
public final class Stopwatch {

    /**
     * Start the stopwatch.
     *
     * @throws IllegalStateException if the stopwatch is already running.
     */
    public void start(){
        if ( fIsRunning ) {
            throw new IllegalStateException("Must stop before calling start
again.");
        }
        //reset both start and stop
        fStart = System.currentTimeMillis();
        fStop = 0;
        fIsRunning = true;
        fHasBeenUsedOnce = true;
    }

    /**
     * Stop the stopwatch.
     *
     * @throws IllegalStateException if the stopwatch is not already running.
     */
    public void stop() {
        if ( !fIsRunning ) {
```

StopWatch.java

```
        throw new IllegalStateException("Cannot stop if not currently running.");
    }
    fStop = System.currentTimeMillis();
    fIsRunning = false;
}

/**
 * Express the "reading" on the stopwatch.
 *
 * @throws IllegalStateException if the Stopwatch has never been used,
 * or if the stopwatch is still running.
 */
public String toString() {
    validateIsReadable();
    StringBuffer result = new StringBuffer();
    result.append(fStop - fStart);
    result.append(" ms");
    return result.toString();
}

// find the time used
public long timeUsed(){
    return fStop - fStart;
}

/**
 * Express the "reading" on the stopwatch as a numeric type.
 *
 * @throws IllegalStateException if the Stopwatch has never been used,
 * or if the stopwatch is still running.
 */
public long toValue() {
    validateIsReadable();
    return fStop - fStart;
}

// PRIVATE ////
private long fStart;
private long fStop;

private boolean fIsRunning;
private boolean fHasBeenUsedOnce;

/**
 * Throws IllegalStateException if the watch has never been started,
 * or if the watch is still running.
 */
private void validateIsReadable() {
    if ( fIsRunning ) {
        String message = "Cannot read a stopwatch which is still running.";
        throw new IllegalStateException(message);
    }
    if ( !fHasBeenUsedOnce ) {
        String message = "Cannot read a stopwatch which has never been
started.";
        throw new IllegalStateException(message);
    }
}
}
```


Glossary

Automatic Composition Engine (ACE)

ACE is a component in SCE. The main functionalities of ACE are to automatically conduct service composition based on formal request. The service compositions constructed by ACE are described in SPATEL which are ready to be executed by SPATEL execution engine.

Context Management Framework (CMF)

CMF is a framework created in SPICE project. An objective of CMF is to provide context handling functionality. CMF comprises of three components, namely, context source, context sink and context broker.

Consolidated service

Consolidated service is 1) a service that has service goal corresponding to user request. or 2) a service that can be composed with other services to satisfy user request.

Context free grammar (CFG)

A context free grammar (CFG) is a grammatical rule that has a form $L \rightarrow R$. An element on the left hand side is called non-terminal element. An element that is existed only on the right hand side is called terminal element.

Constant

In term of NLP, constant is an object in FOPC. It is used to represent an identifiable object in the sentence such as a restaurant name.

Context Source

Context Source is a component in CMF. It is used to publish context (knowledge).

Context Sink

Context Sink is a component in CMF. It is used to subscribe/retrieve a context provided by Context Source.

Context Broker

Context Broker is a component in CMF. It provides functionality to register Context Source and facilitate Context Sink to search for the source in order to obtain a particular type of knowledge.

Dictionary file

In term of IA, dictionary file is a database utilized by IA. It is used by IA to translate Norwegian word to English word.

End node

End node is a service component that does not provide input to other service components.

Elementary predications (EP)

In term of MRS, EP is a basis structural unit. It indicates a single semantic relation with its required arguments.

Fundamental feature

Fundamental feature is a feature that constitutes to a core functionality of IA. Fundamental features are abilities for IA to provide simple interface to the end user, to subscribe to user contexts (by implementing context sink), to extract keywords from user request, to retrieve concepts (e.g. relationship between service and keywords) that are relevant to the keywords from ontology files and to discover a list of services.

Function

In term of NLP, function is an object in FOPC. It is used to represent an object that forms a connection with other objects such as a genitive, a location of the restaurant. An example of function is LocationOf(Maharani). It looks like predicate but it is in fact term because it refers to unique object.

Functional property

Functional property is a component in a request sent to ACE. It describes a characteristic of service composition.

Formal request

In term of ACE, formal request is a request created by service developer. The request consists of semantic information necessary for describing service composition.

First Order Predicate Calculus (FOPC)

In the scope of this thesis, FOPC is an approach to represent a meaning of sentence (i.e. a tool for semantic representation). The most attractive feature of FOPC is its flexibility to represent complex sentences.

Goal ontology

Goal ontology is an ontology file that is used to store semantics of service goals and relationship between service goals.

Grammatical element

Grammatical element is a synonym of syntactical element.

Head-Driven Phrase Structure Grammar (HPSG)

HPSG is a framework that is used to capture a grammatical structure of sentence resulted from a parsing process.

Handle

In term of MRS, handle is a node of MRS tree which identifies an EP with a potential scopal argument position.

Intelligent Agent (IA)

IA is an application developed in this thesis. It provides an extension to SCE. It supports two main features. The two features are natural language processing and context handling. The two features are important for its service discovery and composition process.

Input/Output ontology

Input/Output ontology is an ontology file that is used to store semantics of service inputs/outputs and relationship between service inputs/outputs

IA module

IA module is a java class that constitutes/interacts with other modules to form IA. Examples of IA module are UIParser and RdfReader.

Information retrieval

Information retrieval [4] is a technique that is used to extract keywords from a sentence. Information retrieval approximates a meaning of sentence from meanings of individual words (in fact, it approximates a meaning of sentence from meanings of “some” keywords in the sentence). Information retrieval does not consider how an interaction of words form a new meaning (i.e. information retrieval ignores preposition such as “and”, “to”, “from” etc).

Informal request

In term of ACE, informal request is a request in natural language created by end user. The informal request is a free-form request that has to be analyzed by IA before service composition can be deduced.

Knowledge Management Framework (KMF)

KMF is a framework created in SPICE project. An objective of KMF is to provide context handling functionality. KMF comprises of three components, namely, knowledge source, knowledge sink and knowledge broker.

Knowledge Source

Knowledge Source is a component in KMF. It is used to publish context (knowledge).

Knowledge Sink

Knowledge Sink is a component in KMF. It is used to subscribe/retrieve a context provided by Knowledge Source.

Knowledge Broker

Knowledge Broker is a component in KMF. It provides functionality to register Knowledge Source and facilitate Knowledge Sink to search for the source in order to obtain a particular type of knowledge.

Minimal Recursive Semantic (MRS) file

MRS file is a file generated by an application (i.e. NLP-tool) that has been developed by linguistic expert. It indicates relationships between words in the user request. It also indicates semantic properties of words in the user request. This file is necessary when IA conducts NLP.

Minimal Recursion Semantic (MRS)

MRS is a semantic framework (that is used for capturing semantic representation of a sentence) designed with principles to support large-scale computational applications of semantic theory.

NLP-tool

NLP-tool is an application developed by linguistic experts. It provides a function to parse and analyze Norwegian sentence. The output of NLP-tool is MRS file.

Natural Language Processing (NLP)

NLP [4] is a subfield of artificial intelligence and computational linguistics. NLP involves two main operations that are 1) automated generation and 2) understanding of natural human languages.

Non-terminal element

In term of CFG, non-terminal element is an element that can be substitute by terminal element.

Non-functional property

Non-functional property is a component in a request sent to ACE. It describes criteria of choosing service compositions. For example, it indicates service value such as what is the cost for the user to run the service.

Ontology concept

Ontology concept is an ontology class (in RDF or OWL) that identifies a resource. Ontology concept can form a relationship with other concepts and constitute into a deeper meaning.

Optional goal

Optional goal is a type of functional property. It indicates a goal that is not mandatory for service composition. This means that the service composition is considered to be completed even if the service composition cannot achieve this goal.

Predicate

In term of NLP, predicate is an object in FOPC. It is used to represent relationships between terms. In term of RDF or OWL, predicate is one of objects in triple <subject, predicate,object> that is used to identify a resource.

Propositional calculus

Propositional calculus is a system of logical formulas that is concerned with expressions (e.g. of sentence) over the Boolean type which has two values, namely, true and false.

Propositional temporal logic

Propositional temporal logic is a type of temporal logic that is extended from propositional calculus. The propositional calculus is extended with two unary temporal operators describing time of event.

Quality of Service (QoS)

QoS is a service parameter indicating a degree in which the service satisfies the user request.

Restriction service rule

Restriction service rule is a database utilized by IA. It contains information about services that should not be invoked once certain user contexts arise.

Recommended service rule

Recommended service rule is a database utilized by IA. It contains information about services that should be invoked once certain user contexts arise.

Resource Description Framework (RDF)

RDF is a framework to describe ontologies which identified the resources (e.g. objects) and their relations on the web.

Service component

Service component is a service that is composed with other services to form a service composition.

SPATEL execution engine

SPATEL execution engine is a component in SEE. The main functionalities of SPATEL execution engine is to take service compositions, that are described in SPATEL, together with the service input parameters as input and invoke the service composition.

Service Creation Environment (SCE)

SCE is a framework created in SPICE project. An objective of SCE is to create and compose heterogeneous web services and Telecommunication services.

Service Execution Environment (SEE)

SEE is a framework created in SPICE project. An objective of SEE is to execute compositions of the services created by SCE.

Start node

In term of ACE, start node is a service component that takes at least one input parameter from the user (and not from other service components). In term of IA, start node is a service component that does not take input parameters from other service components.

Service composition

Service composition is a process of combining a set of different simple services (e.g. web services, telecommunication services) to accomplish a larger and sophisticated task or business process to satisfy user requirement.

Service goal

Service goal is an aim of service. It indicates an intention of service provider to achieve some tasks.

Service operation

Service operation is a function that a service provides to the user. Technically, it is a name of function call (e.g. as in traditional programming language) that the user of service invokes.

Service input

Service input is a type of parameter that the service operation takes. An example of service input is *CityName*.

Service input instance

Service input instance is a parameter that the service operation takes. An example of service input instance is *Paris*.

Service input parameter

Service input parameter is a synonym of service input instance.

Service output

Service output is a type of the result obtained from invoking the service operation.

Service repository

Service repository is a repository storing a list of services available to the user.

SPATEL request

SPATEL request is a request of IA to ACE. The request contains information about service composition that will be converted to SPATEL by ACE.

Synonym ontology

Synonym ontology is an ontology file that is used to store dependency between words that share the same meaning.

Speech request

Speech request is a request in spoken natural language that human provides to machine.

Sentence element

Sentence element is an element that refers to entire sentence. It covers all other grammatical elements such as noun phrase and verb phrase.

Semantic analysis

Semantic analysis [4] is a process to derive and transform a meaning of sentence (in natural language) in to a form that machine understand.

Syntactical element

Syntactical element is a node in a parse tree generated by a natural language parser.

Semantic component

Semantic component is a grammatical element that is attached with semantic (based on semantic attachment rules).

Semantic representation

Semantic representation is a process to represent a meaning of a sentence (e.g. user request) in a form that machine is understand.

SPICE Advance Language for Telecommunication Services (SPATEL)

SPATEL is a high-level language developed in SPICE project. It is used for designing service compositions.

Textual request

Textual request is a request in written natural language that human provides to machine.

Terminal element

In term of CFG, terminal element is an element that cannot be deduced further using CFG rule.

Term

In term of NLP, Term is an object in FOPC. It is used to represent objects (e.g. words) in a sentence. Term can be classified into constants, functions or variables.

Temporal logic

Temporal logic is a system of logical formulas that is used to reason about time of event.

The Web Ontology Language (OWL)

OWL is a machine interpretable language that extends RDF capability. OWL has three main components: Individuals, Properties and Classes.

User request

User request is a textual request from the end user to IA. It is an English sentence.

Variable

In term of NLP, variable is an object in FOPC. It is used to represent an anonymous object or all objects in the domain (i.e. sentence).

References

- [1] Johan Iboma, “Study Shows Half the World’s Population with Mobile Phones by 2008”. December 2007. [Online]. Available: <http://www.nextbillion.net/newsroom/2007/12/06/study-shows-half-the-worlds-population-with-mobile-phones-by-2008>, last access: June 24, 2008
- [2] Christophe Cordier, Francois Carrez, Herma Van Kranenburg, Carlo Licciardi, Jan Van der Meer, Antonietta Spedalieri, Jean-Pierre Le Rouzic, Josip Zoric, “Addressing the challenges of Beyond 3G service delivery: the SPICE service platform”, **6th International Workshop on Applications and Services in Wireless Networks (ASWN 2006)**, May 2006, Berlin Germany, pp. 1-29
- [3] Mazen Shiaa, Paolo Falcarin, Alain Pastor, Freddy Lecue, Eduardo Silva, Luis Ferreira Pires, Mariano Belaunde, “Simplifying Automatic Service Composition – End-user and Service developer perspectives”, February 2008, IEEE transactions on service computing pp 1-12
- [4] Daniel Jurafsky and James H. Martin, “SPEECH and Language Processing, An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition”, International Edition, February 2002, **Prentice Hall**
- [5] Julien Pauty, Davy Preuveneers, Peter Rigole, Yolande Berbers, “Research Challenges in Mobile and Context-Aware Service Development”, **Workshop on Research Challenges in Mobile and Context-Aware Service Development, FCSS 2006**, April 2006, Vienna Austria, pp. 141-148
- [6] Christophe Cordier, Herma van Kranenburg, “Specification of the Knowledge Management Framework”, **The SPICE project**, January 2006, pp 1-49
- [7] Ivar Jorstad, Do van Thanh, Schahram Dustdar, “An Analysis of Service Continuity in Mobile Service”, **13th IEEE International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises**, June 2004, pp 121-126
- [8] Amir Tomer, “Iterative Software Development- from Theory to Practice”, **1st International Conference on Software Management**, February 2002, Anaheim, CA
- [9] Mordechai Ben-Ari, “Mathematical Logic for Computer Science”, 2nd Edition, 2001, **Springer**
- [10] John E. Hopcroft, Rajeev Motwani, Jeffrey D. Ullman, “Automata Theory, Languages and Computation”, International Edition (3rd edition), 2006, **Pearson Education**
- [11] Berners-Lee, Tim; James Hendler and Ora Lassila, “The Semantic Web”, **Scientific American Magazine**, May 2001

- [12] Michael K. Smith, Chris Welly, Deborah L. McGuinness, “W3C OWL Web Ontology Language Guide”, **W3C Recommendation, 10 February 2004**. [Online]. Available: <http://www.w3.org/TR/owl-guide/>, last access: June 24, 2008
- [13] Matthew Horridge, “An Introduction to RDF(S) and a Quick Tour of OWL”, August 2004. [Online]. Available: <http://co-ode.man.ac.uk/resources/tutorials/intro/slides/OWLFoundationsSlides.pdf>, last access: June 24, 2008
- [14] Uche Ogbuiji, “Tip: User rdf:about and rdf:ID effectively with RDF/XML”, February 2003. [Online]. Available: <http://www.ibm.com/developerworks/xml/library/x-tiprdfai.html>, last access: 24 June 24, 2008
- [15] Natalya F. Noy and Deborah L. McGuinness, “Ontology Development 101: A Guide to Creating Your First Ontology”, Online, 2001. [Online]. Available: <http://www.ksl.stanford.edu/people/dlm/papers/ontology101/ontology101-noy-mcguinness.html>, last access: June 24, 2008
- [16] A. Rector, N. Drummond, M. Horridge, J. Rogers, H. Knublauch, R. Stevens, H. Wang, C. Wroe, “OWL Pizzas: Practical Experience of Teaching OWL-DL: Common Errors & Common Patterns”, **European Conference on Knowledge Acquisition (EKAW-2004)**, pp. 63-81.
- [17] Alan Rector, “A Practical Introduction to Ontologies & OWL”, May 2004. [Online]. Available: <http://co-ode.man.ac.uk/resources/tutorials/intro/slides/OWLFoundationsSlides.pdf>, last access: June 24, 2008
- [18] T.R. Gruber, “Towards principles for the design of ontologies used for knowledge sharing,” **Formal Ontology in Conceptual Analysis and Knowledge Representation**, N. Guarino and R. Poli, Eds, 1993, Kluwer Academic Publishers, Deventer, The Netherlands
- [19] Matthew Horridge, Holger Knublauch, Alan Rector, Robert Stevens, Chris Wroe, “A Practical Guide To Building OWL Ontologies Using the Protégé-OWL Plugin and Co-ODE Tools Edition 1.0”, August 2004. [Online]. Available: <http://www.co-ode.org/resources/tutorials/ProtegeOWLTutorial.pdf>, last access: June 24, 2008
- [20] Philip McCarthy, “Introduction to Jena”, June 2004. [Online]. Available: <http://www.ibm.com/developerworks/xml/library/j-jena/>, last access: June 24, 2008
- [21] Philip McCarthy, “Search RDF Data with SPARQL”, May 2005. [Online]. Available: <http://www-128.ibm.com/developerworks/library/j-sparql/>, last access: June 24, 2008
- [22] Leigh Dodds, “Introducing SPARQL: Querying Semantic Web”, November 2005. [Online]. Available: <http://www.xml.com/pub/a/2005/11/16/introducing-sparql-querying-semantic-web-tutorial.html>, last access: June 24, 2008
- [23] Anna V. Zhdanova, Josip Zoric, Marco Marengo, Herma van Kranenburg, Niels Snoeck, Michael Suterer, Christian Rack, Olaf Droegehorn, Stefan Arbanowski, “Context Acquisition, Representation and Employment in Mobile Service Platforms”, **Mobile IST Summit 2006 Workshop on “Capturing Context and Context Aware Systems and Platforms”**, June 2006
- [24] Thibaud Flury, Gilles Privat, Fano Ramparany, “OWL-based location ontology for context-aware ser-

- vices”, **Artificial Intelligence in Mobile Systems**, Nottingham, UK, 2004, pp 1-7
- [25] Robin Sharp, “Principles of Protocol Design”, Draft second edition, 2006, **Faculty of Informatics and Mathematical Modeling, Technical University of Denmark**
- [26] Wade Trappe, Lawrence Washington, “Introduction to Cryptography with Coding Theory”, International edition (2nd edition), 2005, **Pearson Prentice Hall**
- [27] Alexandre V. Smirnov, Mikhail P. Pashkin, Nikolai G. Chilov, Tatiana V. Levashova, Andrew A. Krizhanovsky, “Free Text User Request Processing in the System “KSNet””, **SPECOM’ 2004: 9th Conference Speech and Computer**, September 2004, Saint-Petersburg, Russia, pp 1-4
- [28] Kurt Englmeier, Javier Pereira, Josiance Mothe, “Choreography of Web Services based on Natural Language Storybooks”, **ACM International Conference Proceeding Series, Vol. 156**, 2006, Fredericton, New Brunswick, Canada, pp. 132-138
- [29] Johannes Heinecke, Farouk Toumanic, “A Natural Language Mediation System for E-commerce applications: an ontology-based approach”, **The 2nd International Semantic Web Conference**, October 2003, Sanibel Island, Florida, USA, pp. 39-50
- [30] Ian Wakeman, David Weir, Bill Keller, Julie Weeds. Tim Owen, “Composing Grid Services through Natural Language”, **Workshop on Ubiquitous Computing and e-Research**, May 2005, 4th UK Ubinet workshop, pp. 1-4
- [31] Raphael Enns, Shantha Ramachandran, “An Eclipse GUI Builder”, February 2004. **[Online]**. Available: <http://www.cs.uanitoba.ca/~eclipse>, pp. 1-58, last access: June 24, 2008
- [32] Tao Ye, Li Xue-Qing, Du Ping, Liang Kan, “A Data Mining Based Pervasive User Requests Prediction Method in e-Learning Systems”, **E-Learning in Industrial Electronics, 2006 International Conference on, vol., no.**, December 2006, pp 40-45
- [33] Ann Bies, Mark Ferguson, Karen Katz, Robert MacIntyre, “Bracketing guidelines for Treebank II style penn Treebank project”, 1995. **[Online]**. Available: <http://www.cis.upenn.edu/treebak/home.html>, last access: June 24, 2008
- [34] Hartwig Gunzer, “Introduction to Web Services”, 2002, Borland Corporation, **[Online]**. Available: <http://www.itwizard.info/technology/webservice/webservices.pdf>, last access: June 24, 2008
- [35] Christophe Cordier, Herma van Kranenburg, “Knowledge Management Framework for developer”, **The SPICE project**, January 2006
- [36] Raphael Enns, Shantha Ramachandran, “An Eclipse SWT Designer”, February 2004. **[Online]**. Available: <http://www.cs.uanitoba.ca/~eclipse>, last access: June 24, 2008
- [37] Jason Baldrige, Tom Morton, “The OpenNLP Tools API v1.3.0”, **[Online]**. Available: <http://opennlp.sourceforge.net/api/index.html>, last access: June 24, 2008
- [38] Copestake, A, D. Flickinger, C. Pollard and I Sag: “Minimal Recursion Semantics. An Introduction.”, **Journal of Research on Language and Computation**, 2005, pp. 281-332

- [39] C. Pollard and I. A. Sag, “Head-Driven Phrase Structure Grammar”, **University of Chicago Press and CSLI Publications**, 1994, Chicago, Illinois
- [40] T. Kiss., “Phrasal typology and the interaction of topicalization, wh-movement and extraposition”, **In the Proceedings of the 9th International Conference on Head-Driven Phrase Structure Grammar**, pp. 109-128.
- [41] A. Herbelot and A. Copestake, “Acquiring Ontological Relationships from Wikipedia Using RMRS”, **In Proceedings of the ISWC 2006 Workshop on Web Content Mining with Human Language Technologies**, 2006 [Online], Available: <http://orestes.ii.uam.es/workshop/12.pdf>, last access: June 24, 2008
- [42] J. P. Almeida, A. Baravaglio, M. Belaunde, P. Falcarin and E. Kovacs, “Service Creation in the SPICE Service platform”, **In the Proceedings of the 17th Wireless World Research Forum Meeting**. (WWRF17), November 2006, Heidelberg, Germany
- [43] M. Belaunde, A. Baravaglio, P. Costa, M. Shiaa, P. Falcarin, A. Bosca, P. Larvet, “Advanced Language for Value added services composition and creation”, **The SPICE project**, August 2006
- [44] E. Christensen, F. Curbera, G. Meredith, S. Weerawarana, “Web Services Description Language (WSDL) 1.1”, **W3C Recommendation**, 15 March 2001. [Online]. Available: <http://www.w3.org/TR/wsdl/>, last access: June 24, 2008
- [45] W3C DOM API documentation, **W3C Recommendation**, 2004. [Online]. Available: <http://www.w3.org/TR/2004/REC-DOM-Level-3-Core-20040407/>, last access: June 24, 2008
- [46] J.Bloch, the Sun Java tutorial, [Online]. Available: <http://java.sun.com/docs/books/tutorial/collections/index.html>, last access: June 24, 2008
- [47] J. M. Cargal, “Discrete Mathematics for Neophytes: Number Theory, Probability, Algorithms, and Other Stuff”, [Online]. Available: www.cargalmathbooks.com/5%20Recursive%20Algorithms.pdf, last access: June 24, 2008
- [48] M. Loy, R. Eckstein, D. Wood, J. Elliott, B. Cole, “Java Swing”, **O’ Reilly**, 2nd edition, 2002
- [49] D. Marshall, “A tutorial on ONC RPC”, Cardiff University. [Online]. <http://www.cs.cf.ac.uk/Dave/C/node33.html>, last access: June 24, 2008
- [50] K. Yue, Y. Ge, S. De-rong, N. Tie-zheng, L. Jian, C. Yu, “An Efficient Grid Service Discovery Mechanism Based on the Locality Principle”, **Wuhan University Journals Press**, 2006, China, pp. 83-87

